EPSON

Ver.3.5

SPEL+

Referenzhandbuch

Rev.1

EM027S925U

SPEL+ Referenzhandbuch

Version 3.5
1. Juni 2001

Copyright © 1997-2002 SEIKO EPSON CORPORATION. Alle Rechte vorbehalten.

GARANTIE

Das Robotersystem sowie alle Optionen werden vor Versand an den Kunden sehr strengen Qualitätskontrollen, Tests und Untersuchungen unterzogen, um sicher zu stellen, dass das System in einwandfreiem Zustand ist und unseren hohen Leistungsanforderungen genügt.

Alle Schäden bzw. Fehlfunktionen, die trotz normaler Betriebsbedingungen und Handhabung entstanden sind, werden innerhalb der normalen Garantiezeit kostenlos repariert. (Bitte informieren Sie sich bei Ihrem regionalen EPSON-Vertrieb über die übliche Garantiezeit.)

Für die Reparatur folgender Schäden muss der Kunde aufkommen (selbst wenn sie innerhalb der Garantiezeit auftreten):

1. Schäden oder Fehlfunktionen, die durch nachlässige Bedienung oder Bedienvorgänge verursacht wurden, die nicht in diesem Handbuch beschrieben sind.

2. Unerlaubte kundenseitige Modifikation oder Demontage.

3. Schäden oder Fehlfunktionen, die durch unerlaubte Einstellungen oder Reparaturversuche verursacht wurden.

4. Durch Naturkatastrophen (wie z.B. Erdbeben, Wasserschäden etc.) hervorgerufene Schäden.

Warnungen, Vorsichtsgebote, Nutzung:

1. Wird der Roboter oder mit ihm verbundene Ausrüstung außerhalb der dafür bestimmten Betriebsbedingungen und Produktspezifikationen betrieben, verfällt der Garantieanspruch.

2. Sollten Sie sich nicht an die in diesem Handbuch niedergelegten Warnungen und Vorsichtsgebote halten, müssen wir die Verantwortung für Fehlfunktionen und Unfälle ablehnen, selbst wenn sie in Verletzungen oder Todesfolge resultieren.

3. Wir können nicht alle möglichen Gefahren und die daraus resultierenden Konsequenzen vorhersehen, weshalb dieses Handbuch den Nutzer nicht vor allen Gefahrmomenten warnen kann.

HINWEIS

- Kein Teil dieses Handbuches darf ohne Genehmigung vervielfältigt oder reproduziert werden.
- Wir behalten uns vor, die in diesem Handbuch enthaltenen Informationen ohne Vorankündigung zu ändern.
- Wir bitten Sie freundlich, uns zu kontaktieren, wenn Sie in diesem Handbuch Fehler finden oder uns einen inhaltsbezogenen Kommentar übermitteln wollen.

WARENZEICHEN

Microsoft ist ein eingetragenes Warenzeichen und Windows und das Windows-Logo sind eingetragene Warenzeichen der Microsoft Corporation.

Andere Marken und Produktnamen sind Warenzeichen oder eingetragene Warenzeichen der jeweiligen Inhaber.

Bitte wenden Sie sich mit Fragen bezüglich dieses Handbuches an:

SPEL$^+$ Referenzhandbuch

SEIKO EPSON CORPORATION
Factory Automation Systems Div.
TEL : 81-266-61-1802
FAX : 81-266-61-1846

SICHERHEITSHINWEISE

Bitte studieren Sie dieses Handbuch und andere damit in Verbindung stehende Handbücher eingehend, wenn Sie diese Software benutzen.
Halten Sie dieses Handbuch zu jedem Zeitpunkt griffbereit.

WARNUNG	■ Dieses Zeichen weist Sie auf die Gefahr schwerster Verletzung oder möglicher Todesfolge hin, wenn diese Anweisungen nicht eingehalten werden.
VORSICHT	■ Dieses Zeichen weist Sie auf Verletzungsgefahr von Personen und möglichen physischen Schaden an Geräten und Betriebsanlagen hin, sollten diese Hinweise missachtet werden.

SERVICE-CENTER

Wenn Reparaturen, Wartungsmaßnahmen oder Neueinstellungen notwendig werden, wenden Sie sich bitte an Ihr EPSON Service-Center.
Halten Sie dabei bitte Informationen zur Software-Version und eine kurze Problembeschreibung bereit.

Sollten an dieser Stelle keine Information bezüglich Ihres Service-Centers angegeben sein, wenden Sie sich bitte an Ihren regionalen EPSON-Vertrieb, der im folgenden Abschnitt LIEFERANTEN aufgeführt ist.

LIEFERANTEN

Japan & Andere

SEIKO EPSON CORPORATION
Suwa Minimi Plant
Factory Automation Systems Div.
1010 Fujimi, Fujimi-machi,
Suwa-gun, Nagano 399-0295
Japan

TEL : 81-266-61-1802
FAX : 81-266-61-1846

Nord- & Süd-Amerika

EPSON AMERICA, INC.
Factory Automation/Robotics
18300 Central Avenue
Carson, CA 90746

TEL : (562) 290-5900
FAX : (562) 290-5999
E-MAIL: info@robots.epson.com

Europa

EPSON DEUTSCHLAND GmbH
Factory Automation Division
Zülpicher Str. 6 D-40546 Düsseldorf

TEL : (++) 49 - 211 - 5603 391 (Vertriebsinnendienst)
FAX : (++) 49 - 211 - 5603 444
E-MAIL: robot.infos@epson.de

Inhaltsverzeichnis

Zusammenfassung der SPEL+ Befehle 1

Systemverwaltungsbefehle 1
Roboter-Steuerungsbefehle 2
Eingangs- / Ausgangs-Befehle 4
Befehle zur Änderung von Koordinaten 5
Befehle zur Programmsteuerung 5
Befehle zur Programmausführung 6
Pseudo-Anweisungen 6
Befehle zur Dateiverwaltung 7
Befehle für Variablen 7
Befehle bezüglich numerischer Werte 8
Zeichenketten-Befehle 9
Logische Operatoren 9
Befehle zur Variablendefinition 10
Mit VB-Guide benutzte Befehle 10
Ethernet E/A Befehle 10
Kraftmessungsbefehle 11
Sicherheitsbefehle 11

SPEL+ Sprachreferenz 12

SPEL+ Fehlermeldungen 465

Glossar 483

Zusammenfassung der SPEL+ Befehle

Im folgenden lesen Sie ein Zusammenfassung der SPEL$^+$ Befehle.

Systemverwaltungsbefehle

Reset	Setzt die Robotersteuerung zurück.
Stat	Gibt den Status der Robotersteuerung aus.
Ver	Zeigt die Systemdaten der Robotersteuerung an.
Date	Stellt das Systemdatum ein.
Time	Stellt die Systemzeit ein.
Date$	Gibt das Systemdatum aus.
Time$	Gibt die Systemzeit als Zeichenkette aus.
Hour	Zeigt die Betriebsstunden der Robotersteuerung an oder gibt sie aus.
Time	Stellt die Systemzeit ein.
Errhist	Zeigt ein Fehlerprotokoll an.

Roboter-Steuerungsbefehle

Calib	Kalibriert die Motoren.
CalPls	Definiert bzw. zeigt die Kalibrierungswerte.
Hofs	Definiert die Offset-Pulse zwischen Encoder-Ursprung und Home-Sensor oder gibt sie aus.
Mcal	Kalibriert Inkremental-Encoder-Roboter.
MCordr	Definiert die Reihenfolge der Achsenbewegung bei der Maschinenkalibrierung bzw. gibt die aktuellen Werte aus.
Mcorg	Berechnet die Parameter für die Maschinenkalibrierung.
MCofs	Definiert die Parameter für die Maschinenkalibrierung bzw. zeigt die aktuellen Parameter an.
HTest	Zeigt die Pulse-Zählung vom Home-Sensor zur Encoder-Z-Phase an bzw. gibt sie aus.
Power	Definiert den Servo-Power-Modus bzw. gibt ihn aus.
Motor	Schaltet die Motoren ein bzw. aus
SFree	Schaltet die angegebene Servoachse frei
SLock	Schaltet die angegebene Servoachse wieder ein (nach SFREE)
Jump	Bewegung zu einem Punkt mittels einer PTP-Bewegung.
Arch	Definiert die Bogenparameter für die JUMP-Bewegung bzw. gibt sie aus.
LimZ	Definiert die Z-Achsenhöhe für die JUMP-Bewegung bzw. zeigt den aktuellen Wert an.
Sense	Definiert und zeigt die Bedingungen an, die zu einem Stop einer mit SENSE gekennzeichneten JUMP-Bewegung über dem Zielpunkt führt.
Js	Gibt den Status eines mit SENSE gekennzeichneten JUMP-Vorganges aus.
Jt	Gibt den Status des zuletzt ausgegebenen JUMP-Befehls für den aktuellen Roboter an.
Go	Roboterbewegung zu einem Punkt mittels einer PTP-Bewegung.
Pass	Führt gleichzeitig vier PTP-Bewegungen aus und bewegt den Manipulatorarm dabei an festgelegten Punkten vorbei (Verschleifen von Hilfspunkten, ohne diese exakt zu überfahren).
Pulse	Bewegt den Roboter zu einer in Pulsen definierten Position.
TGo	Führt eine relative PTP-Bewegung im aktuellen Werkzeug-Koordinatensystem aus.
TMove	Führt im gewählten Werkzeug-Koordinatensystem eine linearinterpolierte Bewegung aus.
Till	Definiert bzw. gibt die Eingabebedingung aus, durch die – wenn erfüllt- die aktuell ausgeführte Bewegung beendet wird.
!…!	Verarbeitet Prozessanweisungen parallel zur Ausführung von Bewegungsbefehlen.

Speed	Definiert bzw. gibt die Geschwindigkeit für eine PTP-Bewegung aus.
Accel	Definiert bzw. gibt die Einstellungen für Beschleunigung und Verzögerung von PTP-Bewegungen aus.
Weight	Definiert das Gewicht der Nutzlast und die Länge des 2. Arms bzw. gibt deren Werte aus.
Inertia	Definiert das Massenträgheitsmoment des Roboterarms bzw.gibt dessen Wert aus.
Arc	Führt eine kreisinterpolierte Bewegung aus.
Move	Führt eine linearinterpolierte Roboterbewegung aus.
SpeedS	Definiert die Geschwindigkeit für interpolierte Befehle bzw. gibt sie aus.
AccelS	Definiert die Einstellungen für Beschleunigung und Verzögerung für interpolierte Bewegungen bzw. gibt sie aus.
Home	Bewegt den Roboter zur benutzerdefinierten Home-Position.
HomeSet	Definiert bzw. gibt die Home-Position aus.
Hordr	Definiert die Reihenfolge der Achsenbewegung bei Ausführung des HOME-Befehls.
InPos	Überprüft, ob sich der Roboter in Position befindet (bewegungslos).
CurPos	Gibt während der Bewegung die aktuelle Position aus.
Pallet	Definiert eine Palette oder gibt einen Palettenpunkt aus.
Fine	Definiert die Grenzwerte für Positionierungsfehler.
Qp	Definiert den Quick-Pause-Status bzw. gibt ihn aus.
JRange	Definiert die Arbeitsbereichsgrenzen für eine Achse bzw. gibt sie aus.
Range	Definiert die Grenzwerte für alle Achsen.
XYLim	Definiert die kartesischen Grenzwerte des Roboter-Arbeitsbereiches bzw. gibt sie aus.
CX	Definiert den Koordinatenwert der X-Achse eines Punktes bzw. gibt ihn aus.
CY	Definiert den Koordinatenwert der Y-Achse eines Punktes bzw. gibt ihn aus.
CZ	Definiert den Koordinatenwert der Z-Achse eines Punktes bzw. gibt ihn aus.
CU	Definiert den Koordinatenwert der U-Achse eines Punktes bzw. gibt ihn aus.
Pls	Gibt den Pulse-Wert einer Achse aus.
Agl	Gibt den Achswinkel der aktuellen Position aus.
PAgl	Gibt den Achswert eines bestimmten Punktes aus.
PPls	Gibt die Pulse-Position einer spezifizierten Achse für einen bestimmten Punkt aus.

Robot	Definiert den aktuellen Roboter bzw. gibt ihn aus.

Eingangs- / Ausgangs-Befehle

On	Schaltet den angegebenen Ausgang ein.
Off	Schaltet den angegebenen Ausgang aus.
Oport	Liest den Status eines Ausganges.
Sw	Gibt den Status des Einganges aus.
In	Liest 8 Eingänge.
InW	Gibt den Status des spezifizierten Eingangs-Wordport aus.
InBCD	Liest 8 Eingänge als BCD-Code (Binär-Dezimal-Code).
Out	Setzt 8 Ausgänge bzw. gibt deren Status aus.
OutW	Setzt 16 Ausgänge gleichzeitig.
OpBCD	Definiert 8 Ausgänge gleichzeitig unter Verwendung des BCD-Formats.
On $	Schaltet einen Merker ein.
Off $	Schaltet einen Merker aus.
Sw($)	Gibt den Status eines Merkers aus.
In($)	Liest 8 Merker.
Out $	Setzt 8 Merker bzw. gibt deren Status aus.
ZeroFlg	Gibt den Status eines Merkers vor der letzten Zustandsänderung aus.
Wait	Wartet auf eine Bedingung oder Zeit.
TMOut	Definiert das Zeitlimit für die WAIT-Anweisung.
Tw	Gibt den Status der WAIT-Bedingung (Zeitüberschreitung) und des WAIT-Timer-Intervalls aus.
Input	Aufforderung zur Eingabe einer oder mehrerer Variablen über das aktuelle Anzeigegerät.
Print	Zeigt Daten auf dem aktuellen Anzeigegerät an.
Line Input	Dateneingabe einer Zeichenkette vom aktuellen Anzeigegerät.
Input #	Aufforderung zur Eingabe einer oder mehrerer Variablen aus einer Datei oder von einem Port.
Print #	Datenausgabe an eine Datei oder einen Port.
Line Input #	Eingabe einer Zeichenkette aus einer Datei oder von einem Port.
Lof	Gibt die Anzahl von Zeilen in einem Kommunikationspuffer aus.
ClrScr	Löscht den *EPSON RC+* Run oder den Bedienerfenster-Textbereich.
EPrint	Beendet eine Drucksequenz von LPrint-Anweisungen und sendet an den Drucker.
InputBox	Zeigt die Eingabebox an und gibt die Anwender-Antwort aus.
LPrint	Sendet eine Zeile zum Standarddrucker.
MsgBox	Zeigt dem Anwender eine Nachricht an.

Befehle zur Änderung von Koordinaten

Arm	Definiert den aktuellen Arm bzw. gibt ihn aus.
ArmSet	Definiert einen Arm.
Tool	Definiert die aktuelle Werkzeugnummer bzw. gibt sie aus.
TLSet	Definiert ein Werkzeugkoordinatensystem oder zeigt es an.
Local	Definiert ein lokales Werkzeugkoordinatensystem.
Base	Definiert das Basis-Koordinatensystem und zeigt es an.

Befehle zur Programmsteuerung

Function	Deklariert eine Funktion.
For...Next	Führt eine oder mehrere Anweisungen in Form einer Schleife bis zum angegebenen Zählwert aus.
GoSub	Führt ein Unterprogramm aus.
Return	Rückkehr zum Hauptprogramm nach Ausführung eines Unterprogramms.
GoTo	Verzweigt ohne Bedingung zu einer Zeilennummer oder zu einem Label.
Call	Ruft eine Anwender-Funktion auf.
If..Then..Else..EndIf	Führt eine Anweisung bedingt aus.
Else	Wird mit der *If*-Anweisung benutzt, damit Anweisungen auch dann ausgeführt werden können, wenn die *If*-Anweisung falsch ist. Else ist eine Option für die If/Then Anweisung.
Select ... Send	Definiert die Verzweigungsformel sowie die entsprechenden Anweisungssequenzen zum Verzweigen.
While...Wend	Führt eine angegebene Anweisung aus, solange eine festgelegte Bedingung erfüllt ist.
Do...Loop	Do...Loop Konstrukt.
Trap	Spezifiziert eine Interrupt-Behandlungsroutine.
OnErr	Definiert eine Fehlerbehandlungsroutine.
Era	Gibt die Roboterachsnummer für den zuletzt aufgetretenen Fehler aus.
Erl	Gibt die Zeilennummer des Fehlers aus.
EClr	Löscht den aktuellen Fehler.
Err	Gibt die Fehlernummer aus.
Ert	Gibt die Tasknummer des Fehlers aus.
ErrMsg$	Gibt die Fehlermeldung aus.

Befehle zur Programmausführung

Xqt	Führt einen Task aus.
Pause	Unterbricht kurzfristig die Programmausführung bei all den Tasks, bei denen der PAUSE-Befehl aktiviert ist.
Cont	Fortfahren nach der Pause.
Halt	Unterbrechen eines Tasks.
Quit	Beendet einen Task.
Resume	Setzt einen im Halt-Status befindlichen Task fort.
MyTask	Gibt den aktuellen Task aus.
Chain	Stoppt alle Tasks und startet eine Programmgruppe.

Pseudo-Anweisungen

#define	Definiert ein Makro.
#ifdef ... #endif	Bedingte Kompilierung.
#ifndef ... #endif	Bedingte Kompilierung.
#include	Einbeziehung einer Datei.

Befehle zur Dateiverwaltung

Dir	Zeigt ein Verzeichnis an.
ChDir	Wechselt das aktuelle Verzeichnis.
MkDir	Erstellt ein Verzeichnis.
Rmdir	Entfernt ein Verzeichnis.
Rendir	Benennt ein Verzeichnis um.
FileDateTime$	Gibt Dateidatum und –zeit aus.
FileExists	Überprüft die Existenz einer Datei.
FileLen	Gibt die Länge einer Datei aus.
FolderExists	Überprüft die Existenz eines Ordners.
LoadPoints	Lädt die Punkte für den aktuellen Roboter.
SavePoints	Speichert die Punkte für den aktuellen Roboter.
Type	Zeigt den Inhalt einer Textdatei an.
Kill	Löscht eine Datei.
Del	Löscht eine Datei.
Copy	Kopiert eine Datei.
Rename	Benennt eine Datei um.
AOpen	Öffnet eine Datei zum Anfügen von Daten.
ROpen	Öffnet eine Datei zum Einlesen von Daten.
WOpen	Öffnet eine Datei zum Schreiben von Daten.
Read	Liest ein oder mehrere Zeichen aus einer Datei oder von einem Port.
Seek	Verschiebt einen Dateizeiger für den Direktzugriff.
Write	Schreibt ein oder mehrere Zeichen in eine Datei oder einen Port.
Close	Schließt eine Datei.

Befehle für Variablen

Globals	Zeigt im Speicher befindliche globale Variablen an.

Befehle bezüglich numerischer Werte

Oport	Liest den Status eines Ausganges.
Sw	Gibt den Status eines Einganges aus.
In	Liest 8 Eingänge.
InBCD	Liest 8 Eingänge als BCD-Code (Binär-Dezimal-Code).
Sw($)	Gibt den Status eines Merkers aus.
In($)	Liest 8 Merker.
ZeroFlg	Gibt den Status eines Merkers vor der letzten Zustandsänderung aus.
Ctr	Gibt den Zählerstand eines Zählers aus.
CTReset	Setzt einen Zähler zurück.
Tmr	Gibt den Zählerstand eines Timers aus.
TmReset	Setzt einen Timer auf 0 zurück.
Time	Stellt die Systemzeit ein.
Js	Gibt den Status eines mit SENSE gekennzeichneten JUMP-Vorganges aus.
CX	Definiert den Koordinatenwert der X-Achse eines Punktes bzw. gibt ihn aus.
CY	Definiert den Koordinatenwert der Y-Achse eines Punktes bzw. gibt ihn aus.
CZ	Definiert den Koordinatenwert der Z-Achse eines Punktes bzw. gibt ihn aus.
CU	Definiert den Koordinatenwert der U-Achse eines Punktes bzw. gibt ihn aus.
Pls	Gibt den Pulse-Wert einer Achse aus.
Agl	Gibt den Achswinkel der aktuellen Position aus.
Era	Gibt die Roboterachsnummer für den zuletzt aufgetretenen Fehler aus.
Erl	Gibt die Zeilennummer des Fehlers aus.
Err	Gibt die Fehlernummer aus.
Ert	Gibt die Tasknummer des Fehlers aus.
Tw	Gibt den Status der WAIT-Bedingung (Zeitüberschreitung) und des WAIT-Timer-Intervalls aus.
MyTask	Gibt den aktuellen Task aus.
Lof	Gibt die Anzahl von Zeilen in einem Kommunikationspuffer aus.
Stat	Gibt den Status der Robotersteuerung aus.
Sin	Gibt den Sinuswert eines Winkels aus.
Cos	Gibt den Kosinuswert eines Winkels aus.
Tan	Gibt den Tangenswert eines Winkels aus.
Atan	Gibt den Arkustangens aus.
Atan2	Gibt den Arkustangens auf der Basis der X,Y –Position aus.
Sqr	Gibt die Quadratwurzel einer Zahl aus.

Abs	Gibt den Absolutwert einer Zahl aus.
Sgn	Gibt das Vorzeichen einer Zahl aus.
Int	Konvertiert eine relle Zahl in einen Integer.
Not	NICHT-Operator.
LShift	Verschiebt Bits nach links.
RShift	Verschiebt Bits nach rechts.

Zeichenketten-Befehle

String	Definiert Zeichenkettenvariablen.
Asc	Gibt den ASCII wert eines Zeichens aus.
Chr$	Gibt das Zeichens eines numerischen ASCII-Wertes aus.
Left$	Gibt eine Teilkette von der linken Seite einer Zeichenkette aus.
Mid$	Gibt eine Teilkette aus.
Right$	Gibt eine Teilkette von der rechten Seite einer Zeichenkette aus.
Len	Gibt die Länge einer Zeichenkette aus.
Space$	Gibt eine Zeichenkette aus, die Leerzeichen enthält.
Str$	Konvertiert eine Zahl in eine Zeichenkette.
Val	Konvertiert eine numerische Zeichenkette in eine Nummer.
ErrMsg$	Gibt die Fehlermeldung aus.
LCase$	Konvertiert eine Zeichenkette in Kleinbuchstaben.
UCase$	Konvertiert eine Zeichenkette in Großbuchstaben.
LTrim$	Entfernt Zeichen von der linken Seite der Zeichenkette.
RTrim$	Entfernt Zeichen von der rechten Seite der Zeichenkette.
ParseStr	Analysiert eine Zeichenkette syntaktisch und gibt eine Textelement-Matrix aus.
FmtStr	Formatiert eine Zahl oder Zeichenkette.

Logische Operatoren

And	Führt eine logische und Bit-weise UND-Operation durch.
Or	ODER-Operator.
LShift	Verschiebt Bits nach links.
Mod	Modulo-Operator.
Not	Negation.
RShift	Verschiebt Bits nach rechts.
Xor	EXKLUSIV-ODER-Operator.

Befehle zur Variablendefinition

Boolean	Deklariert Boolesche Variablen.
Byte	Deklariert Byte-Variablen.
Double	Deklariert doppelte Variablen.
Integer	Deklariert Integer Variablen.
Long	Deklariert Long Integer Variablen.
Real	Deklariert Real Variablen.
String	Deklariert Zeichenkettenvariablen.

Mit VB-Guide benutzte Befehle

SPELCom_Event	Löst ein Event im ActiveX Client aus.
SPELCom_Return	Legt den Funktionsrückgabewert für den ActiveX Client fest.

Ethernet E/A Befehle

ENetIO_AnaGetConfig	Gibt die Konfigurationswerte des analogen Eingabe- oder Ausgabemoduls aus.
ENetIO_AnaIn	Liest einen analogen Eingabe- oder Ausgabekanal.
ENetIO_AnaOut	Stellt den Wert für einen analogen Ausgang ein.
ENetIO_AnaSetConfig	Konfiguriert ein analoges Eingabe- oder Ausgabemodul.
ENetIO_ClearLatches	Löscht sowohl das digitale On- als auch das digitale Off-Latch für einen Eingangspunkt.
ENetIO_In	Liest den Wert eines digitalen Eingabe- oder Ausgabemoduls (4 Punkte).
ENetIO_Off	Schaltet einen digitalen Ausgang aus.
ENetIO_On	Schaltet einen digitalen Ausgang ein.
ENetIO_Oport	Gibt den Status eines digitalen Ausganges aus.
ENetIO_Out	Setzt die Ausgänge eines digitalen Ausgabemoduls (4 Punkte).
ENetIO_Sw	Gibt den Status eines digitalen Einganges aus.
ENetIO_SwLatch	Gibt den Latch-Status eines digitalen Eingangs aus.

Kraftmessungsbefehle

Force_Calibrate	Setzt alle Achsen des aktuellen Sensors auf Null.
Force_ClearTrigger	Löscht alle Auslösebedingungen für den aktuellen Sensor.
Force_GetForce	Gibt den aktuellen Wert für eine Achse des aktuellen Sensors aus.
Force_GetForces	Gibt die aktuellen Werte für alle Achsen des aktuellen Sensors in einer Matrix aus.
Force_Sensor	Stellt den aktuellen Sensor für den aktuellen Task ein bzw. gibt ihn aus.
Force_SetTrigger	Stellt den Kraftbegrenzungsauslöser des aktuellen Sensors für Bewegungsbefehle ein.
Force_TC	Stellt den aktuellen Modus der Drehmomentsteuerung für den aktuellen Roboter ein bzw. gibt ihn aus.
Force_TCLim	Stellt die aktuellen Drehmoment-Begrenzungen für den aktuellen Roboter ein bzw. gibt sie aus.
Force_TCSpeed	Stellt die aktuelle Geschwindigkeit nach Erreichen der Drehmoment-Begrenzungen für den aktuellen Roboter ein bzw. gibt sie aus.

Sicherheitsbefehle

GetCurrentUser$	Gibt den aktuellen EPSON RC+ Anwender aus.
Login	Loggt als anderer Anwender in EPSON RC+ ein.

SPEL+ Sprachreferenz

Dieser Teil beschreibt jeden SPEL$^+$ -Befehl in folgenderArt und Weise:

Syntax	Die Syntax beschreibt das Format, das für jeden Befehl benutzt wird. Für einige Befehle wird mehr als eine Syntax angezeigt, zusammen mit einer Zahl, auf die in der Befehlsbeschreibung Bezug genommen wird. Parameter werden in Kursivschrift dargestellt.
Parameter	Beschreibt jeden in diesem Befehl enthaltenen Parameter.
Rückgabewerte	Beschreibt alle Werte, die auf den Befehl hin ausgegeben werden.
Beschreibung	Beschreibt Details hinsichtlich der Funktionsweise des Befehls.
Einschränkung	Zeigt potentielle Probleme auf, wenn dieser Befehl in bestimmten Situationen genutzt wird.
Hinweise	Gibt zusätzliche Informationen, die hinsichtlich dieses Befehls wichtig sein können.
Verwandte Befehle	Zeigt andere, mit diesem Befehl zusammenhängende Befehle. Schauen Sie bitte im Inhaltsverzeichnis nach der Seitennummer dieser verwandten Befehle.
Beispiel	Gibt ein oder mehrere Beispiele, wie dieser Befehl benutzt wird.

SYMBOLE

Dieses Handbuch benutzt die folgenden Symbole um zu zeigen, in welchem Kontext dieser Befehl benutzt werden kann:

>	Kann als Online-Modus-Befehl verwendet werden.
S	Kann als Anweisung in einem SPEL$^+$ Programm verwendet werden.
F	Funktion
INC	Bestimmt für INC-Roboter ausgestattet mit inkrementellem Encoder. Wird ein so gekennzeichneter Befehl bei einem ABS-Roboter mit absolutem Encoder ausgeführt, tritt Fehler 123 auf.

!...! Parallelbearbeitung

> S

Verarbeitet Ein-/Ausgangsanweisungen parallel zur Ausführung von Bewegungsbefehlen.

Syntax

motion cmd **!***statements* **!**

Parameter

motion cmd — Jeder gültige Bewegungsbefehl der folgenden Liste: Jump, Go, Move, Arc oder Pulse.

statements — Alle gültigen Parallelbearbeitungs- E/A-Anweisung(en), die während der Bewegung ausgeführt werden können. (Siehe unten stehende Tabelle)

Beschreibung

Parallelbearbeitungsbefehle sind Bewegungsbefehlen angegliedert, damit E/A-Anweisungen gleichzeitig mit dem Beginn der Verfahrbewegung ausgeführt werden können. Dies bedeutet, dass E/A ausgeführt werden kann, *während* sich der Arm bewegt, anstatt zu warten, bis der Arm eine Stop-Position erreicht hat und E/A *dann* auszuführen. Ebenfalls gibt es eine Einrichtung, welche definiert, zu welchem Zeitpunkt innerhalb der Bewegung die Ausführung von E/A beginnen sollte. (Siehe auch die in der unten stehenden Tabelle beschriebenen Dn-Parameter.)

Die folgende Tabelle zeigt alle gültigen Parallelbearbeitungsbefehle. Jeder dieser Befehle kann als Einzelbefehl verwendet werden, oder zusammen in einer Guppe, damit während einer Bewegungsanweisung mehrfache E/A-Anweisungen ausgeführt werden können.

Dn	- Legt das Timing für die Ausführung der Parallelbearbeitung fest. n ist eine Real-Zahl zwischen 0 und 100, welche die Position innerhalb der Bewegung repräsentiert, an der die Parallelbearbeitungsbefehle (die den Dn-Parametern folgen) beginnen sollten. Anweisungen welche den Dn-Parametern folgen, beginnen mit der Ausführung, wenn n% der Verfahrbewegung zurückgelegt worden sind.
	- Beim JUMP-Befehl ist die Vertikalbewegung des Z-Achse nicht Bestandteil des %Verfahrwegs. Um Anweisungen auszuführen, nachdem die vertikale Hubbewegung beendet ist, schließen Sie D0 (null) am Beginn der Anweisung mit ein.
	- Dn kann in einer Parallelbearbeitungsanweisung maximal 10 Mal erscheinen.
On/Off n	Schaltet Ausgang Nr. n ein bzw. aus.
On/Off $n	Schaltet Merker Nr. n ein bzw. aus.
Out p,d	Gibt Ausgabedaten d an Ausgangsport p aus.
Out $p, d	Gibt Ausgabedaten d an Merkerport p aus.
Wait t	Verzögert die Ausführung der nächsten Parallelbearbeitungsanweisung um t Sekunden.
Wait Sw(n)=j	Verzögert die Ausführung der nächsten Parallelbearbeitungsanweisung, bis der Eingang n gleich der durch j definierten Bedingung ist. (1=ein; 0=aus)

Wait Sw($n)=j	Verzögert die Ausführung der nächsten Parallelbearbeitungsanweisung, bis der Merker n gleich der durch j definierten Bedingung ist. (1=ein; 0=aus)
Print/Input	Gibt Daten an die Bedieneinheit aus bzw. von dort ein.
Print #/Input #	Gibt Daten an die Kommunikationsschnittstelle aus bzw. von dort ein.

Einschränkungen

Bewegung ist beendet bevor alle E/A-Befehle ausgeführt sind

Falls die Ausführung der Parallelbearbeitungsanweisungen nach Ausführung der Bewegungsbefehle noch nicht abgeschlossen ist, wird die nachfolgende Programmausführung solange verschoben, bis alle Parallelbearbeitungsanweisungen vollständig ausgeführt wurden. Diese Situation tritt mit hoher Wahrscheinlichkeit bei kurzen Bewegungen auf, bei denen parallel viele E/A-Anweisungen ausgeführt werden.

Was geschieht mit der parallelen E/A-Ausführung, wenn die TILL-Anweisung verwendet wird, um den Arm anzuhalten, bevor die beabsichtigte Bewegung vollständig ausgeführt wurde?

Wenn TILL verwendet wird, um den Arm an einer Zwischenposition des Verfahrweges anzuhalten, wird die weitere Programmausführung solange verzögert, bis alle Parallelbearbeitungsanweisungen vollständig ausgeführt wurden.

Hinweise

JUMP-Anweisung und Parallelbearbeitung

Sollten Parallelbearbeitungsanweisungen ausgeführt werden, die zusammen mit dem JUMP-Befehl verwendet werden, so beginnen diese, nachdem die vertikale Hubbewegung vollständig ausgeführt wurde und enden am Beginn der abfallenden Bewegung.

Verwandte Befehle

Arc, Go, Jump, Move, Pulse

!...! Beispiel der Parallelbearbeitung

Das folgende Beispiele zeigt unterschiedliche Arten und Weisen, wie das Parallelbearbeitungsfeature im Zusammenhang mit Bewegungsbefehlen verwendet werden kann:

Parallelbearbeitung mit dem JUMP-Befehl schaltet Ausgang 1 am Ende der Aufwärtsbewegung der Z-Achse ein, und zwar dann, wenn sich die erste, zweite und vierte Achse zu bewegen beginnen. Ausgang 1 wird dann wieder ausgeschaltet, wenn 50% der JUMP-Verfahrbewegung ausgeführt worden sind.

```
Function test
   Jump P1 !D0;On 1;D50;Off 1!
Fend
```

Parallelbearbeitung mit dem JUMP-Befehl schaltet Ausgang 5 ein, nachdem die sich hebende Z-Achse ihre Bewegung beendet hat und wenn das erste, zweite und vierte Achse 10% ihrer Bewegung zum Punkt P1 zurückgelegt haben. o,5 Sekunden später schaltet der Ausgang 5 aus.

```
Function test2
   Move P1 !D10;On 5;Wait 0.5;Off 5!
Fend
```

#define

S

Definiert eine ID-Zeichenkette, die durch eine festgelegte Ersatzzeichenkette ausgetauscht werden soll.

Syntax

#define *identifier* [(*parameter*, [*parameter*])] *string*

Parameter

Hinweise zur ID-Zeichenkette Vom Anwender definiertes Schlüsselwort, welches eine Abkürzung für den *Zeichenketten* Parameter ist. Die Regeln für Zeichenketten lauten wie folgt:

- Das erste Zeichen muss ein Alphabetzeichen, die übrigen können alphanumerisch oder Unterstriche (_) sein.
- Leerzeichen oder Tabulatoren sind als Teil einer *ID-Zeichenkette* nicht gestattet.

Hinweise zu den Parametern Werden normalerweise benutzt, um eine (oder mehrere) Variable(n) zu spezifizieren, die von der Ersatzzeichenkette verwendet werden können. Dies sorgt für einen dynamischen define-Mechanismus, der wie ein Makro benutzt werden kann. Für den #define-Befehl dürfen maximal 8 Parameter verwendet werden. Die einzelnen Parameter müssen jedoch durch Kommata voneinander getrennt werden und die Parameterliste muss in Klammern stehen.

Hinweise zur Zeichenkette Dies ist die Ersatzzeichenkette, welche die ID-Zeichenkette ersetzt, wenn das Programm kompiliert wird. Die Regeln für Ersatzzeichenketten lauten wie folgt:

- Leerzeichen oder Tabulatoren sind als Teil einer Ersatzzeichenkette nicht gestattet.
- ID-Zeichenketten, die mit anderen #define-Anweisungen verwendet werden, können nicht als Ersatzzeichenketten genutzt werden.
- Wird das Kommentarsymbol (') ebenfalls eingebracht, werden die Zeichen nach dem Kommentarsymbol als Kommentar interpretiert und nicht in die Ersatzzeichenkette einbezogen.
- Die Ersatzzeichenkette ist nicht unbedingt erforderlich. In diesem Falle wird die angegebene Zeichenkette durch nichts, oder eine Null-Zeichenkette ersetzt. Dies löscht die ID-Zeichenkette aus dem Programm.

Beschreibung

Der #define-Befehl ersetzt innerhalb eines Programms eine bestimmte ID-Zeichenkette durch eine Ersatzzeichenkette. Jedes Mal, wenn die bestimmte ID-Zeichenkette gefunden wird, wird diese vor dem Kompilieren durch die Ersatzzeichenkette ersetzt. Jedoch verbleibt die ID-Zeichenkette im Quellcode und nicht die Ersatzzeichenkette. Dies macht den Code vielfach leichter lesbar, weil an Stelle schwer lesbarer Code-Zeichenketten aussagekräftige Namen für ID-Zeichenketten verwendet werden.

Die bestimmte ID-Zeichenkette kann für eine bedingte Kompilierung verwendet werden, indem man sie mit den Befehlen #ifdef oder #ifndef kombiniert.

Wenn ein bestimmter Parameter angegeben ist, kann die neue ID-Zeichenkette als Makro verwendet werden.

Einschränkung

Verwenden von #define für die Variablendeklaration oder Labelersetzungen ruft einen Fehler hervor:

Bitte beachten Sie, dass die Verwendung des #define-Befehls für die Variablendeklaration einen Fehler hervorrufen kann.

Verwandte Befehle

#ifdef
#ifndef

#define Beispiel

```
' Unkommentierte nächste Zeile für den Debug-Modus.
' #define DEBUG

Input #1, A$
#ifdef DEBUG
    Print "A$ = ", A$
#endif
Print "The End"

#define SHOWVAL(x) Print "var = ", x

Integer a

a = 25

SHOWVAL(a)
```

#ifdef...#else...#endif

S

if define (falls definiert)

if not define (falls nicht definiert)

Bedingte Kompilierung.

Syntax

#ifdef *[ID-Zeichenkette]*
... schreiben Sie den zur bedingten Kompilierung ausgewählten Quellcode an diese Stelle.
[**#else**
... schreiben Sie den ausgewählten Quellcode für die falsche Bedingung hierher.]
#endif

Parameter

Hinweise zur ID-Zeichenkette Anwenderdefiniertes Schlüsselwort welches, so es definiert ist, es gestattet, dass der zwischen #ifdef und #else oder #endif befindliche Quellcode kompiliert wird. Daher agiert die ID-Zeichenkette als Bedingung für die bedingte Kompilierung.

Beschreibung

#ifdef...#else...#endif gestattet bedingtes Kompilieren eines ausgewählten Quellcodes. Die Bedingung, ob diese Kompilierung durchgeführt wird, oder nicht, hängt von der *ID-Zeichenkette* ab. #ifdef prüft zuerst, ob die angegebene ID-Zeichenkette derzeit von #define definiert wird. Die #else –Anweisung ist nicht unbedingt erforderlich.

Wenn definiert und die #else –Anweisung nicht genutzt wird, werden die Anweisungen zwischen #ifdef und #endif kompiliert. Anderenfalls, d.h. sollte die #else –Anweisung genutzt werden, werden die Anweisungen zwischen #ifdef und #else kompiliert.

Wenn nicht definiert und die #else –Anweisung nicht genutzt wird, werden die Anweisungen zwischen #ifdef und #endif ohne Kompilierung übersprungen. Anderenfalls, d.h. sollte die #else –Anweisung genutzt werden, werden die Anweisungen zwischen #else und #endif kompiliert.

Verwandte Befehle

#define, #ifndef

#ifdef Beispiel

Ein Abschnitt des Codes aus einem Beispielprogramm, welches #ifdef verwendet, ist unten abgebildet. Im unten stehenden Beispiel erfolgt ein Ausdruck des A$-Variablenwertes, je nachdem, ob eine Definition des #define DEBUG Pseudo-Befehls vorliegt oder nicht. Wenn der #define DEBUG Pseudo-Befehl in dieser Quelle bereits im Vorangegangenen benutzt wurde, so wird die Zeile ‚Print A$' kompiliert und später ausgeführt, wenn das Programm läuft. Der Ausdruck der Zeichenkette "The End" jedoch, wird trotzt des #define DEBUG Pseudo-Befehls erfolgen.

```
'Unkommentierte nächste Zeile für den Debug-Modus.
' #define DEBUG

Input #1, A$
#ifdef DEBUG
    Print "A$ = ", A$
#endif
Print "The End"
```

#ifndef...#endif

S

Bedingte Kompilierung.

Syntax

#ifndef *[ID-Zeichenkette]*
... schreiben Sie den zur bedingten Kompilierung ausgewählten Quellcode an diese Stelle.
[**#else**
... schreiben Sie den ausgewählten Quellcode für die wahre Bedingung hierher.]
#endif

Parameter

Hinweise zur ID-Zeichenkette Anwenderdefiniertes Schlüsselwort welches, wenn es **nicht** definiert ist, gestattet, dass der zwischen #ifndef und #else oder #endif befindliche Quellcode kompiliert wird. Daher agiert die ID-Zeichenkette als Bedingung für die bedingte Kompilierung.

Beschreibung

Dieser Befehl wird "if not defined" (“falls nicht definiert”) –Befehl genannt. #ifndef...#else...#endif gestattet bedingtes Kompilieren eines ausgewählten Quellcodes. Die #else –Anweisung ist nicht unbedingt erforderlich.

Wenn definiert und die #else –Anweisung nicht genutzt wird, werden die Anweisungen zwischen #ifndef und #endif nicht kompiliert. Anderenfalls, d.h. sollte die #else –Anweisung genutzt werden, werden die Anweisungen zwischen #else und #endif kompiliert.

Wenn nicht definiert und die #else –Anweisung nicht genutzt wird, werden die Anweisungen zwischen #ifndef und #endif kompiliert. Anderenfalls, d.h. sollte die #else –Anweisung genutzt werden, werden die Anweisungen zwischen #else und #endif nicht kompiliert.

Hinweis:

Der Unterschied zwischen #ifdef und #ifndef:

Der grundlegende Unterschied zwischen #ifdef und #ifndef ist, dass der #ifdef –Befehl den angegebenen Quellcode kompiliert, wenn die ID-Zeichenkette **definiert ist**. Der #ifndef –Befehl kompiliert den angegebenen Quellcode, wenn die ID-Zeichenkette **NICHT definiert ist**.

Verwandte Befehle

#define, #ifdef

#ifndef Beispiel

Ein Abschnitt des Codes aus einem Beispielprogramm, welches #ifndef verwendet, ist unten abgebildet. Im unten stehenden Beispiel erfolgt ein Ausdruck des A$-Variablenwertes, je nachdem, ob eine Definition des #define NODELAY Pseudo-Befehls vorliegt oder nicht. Wenn der #define NODELAY Pseudo-Befehl in dieser Quelle bereits im Vorangegangenen benutzt wurde, so wird die Zeile ‘Wait 1’ **NICHT** zusammen mit dem Rest der Quelle für dieses Programm **kompiliert werden**, wenn dieses kompiliert wird. (D.h. zur Ausführung vorgelegt) Wenn der #define NODELAY Pseudo-Befehl in dieser Quelle im Vorangegangenen nicht benutzt wurde (d.h. NODELAY ist nicht definiert), **so wird die Zeile ‚Wait 1’ kompiliert** und später ausgeführt, wenn das Programm läuft. Der Ausdruck der Zeichenkette "The End" wird trotzt des #define NODELAY Pseudo-Befehls erfolgen.

```
' Kommentiert die nächste Zeile aus, um Verzögerungen zu erzwingen.
#define NODELAY 1

Input #1, A$
#ifndef NODELAY
    Wait 1
#endif
Print "The End"
```

#include

S

(Einbeziehen)

Bezieht die angegebene Datei in jene Datei mit ein, welche die #include-Anweisung verwendet.

Syntax

#include "*Dateiname*.**INC"**

Parameter

Hinweise zum Dateinamen Der Dateiname muss der Name einer Include-Datei im aktuellen Projekt sein. Alle Include-Dateien haben eine **INC**-Erweiterung. Der Dateiname spezifiziert die Datei, welche in die aktuelle Datei einbezogen wird.

Beschreibung

#include fügt den Inhalt der spezifizierten Include-Datei in die aktuelle Datei ein, in welcher die #include-Anweisung verwendet wird.

Normalerweise enthalten Include-Dateien #define-Anweisungen.

Eine #include-Anweisung muss außerhalb jeglicher Funktionsdefinition verwendet werden.

Wenn der Pfadname ausgelassen wird, sucht #include nach einer Include-Datei aus dem aktuellen Verzeichnis.

Ein Include-Datei kann eine sekundäre Include-Datei beinhalten. Beispielsweise kann FILE2 in FILE1 enthalten sein und FILE3 in FILE2. Dieser Vorgang wird als Verschachtelung bezeichnet.

Zeilennummern in einzubeziehenden Dateien sind nicht gestattet.

Verwandte Befehle

#define, #ifdef, #ifndef

#include Beispiel

Include-Datei (Defs.inc)

```
#define DEBUG 1
#define MAX_PART_COUNT 20
```

Programmdatei (main.prg)

```
#include "defs.inc"

Function main
    Integer i

    Integer Parts(MAX_PART_COUNT)

Fend
```

ABS-Funktion

F

Absolut

Gibt den Absolutwert einer Zahl aus.

Syntax

Abs(*Zahl*)

Parameter

Zahl Jeglicher gültige numerische Ausdruck.

Rückgabewerte

Der Absolutwert einer Zahl.

Beschreibung

Der Absolutwert einer Zahl ist seine vorzeichenlose Größe. Beispielsweise ist das Ergebnis von **Abs**(-1) und **Abs**(1) in beiden Fällen 1.

Verwandte Befehle

Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Abs Funktionsbeispiel

Die folgenden Beispiele werden mittels des Print-Befehls vom Online-Fenster aus getätigt.

```
> print abs(1)
1
> print abs(-1)
1
> print abs(-3.54)
3.54
>
```

Accel-Anweisung

> S

Acceleration (Beschleunigung)

Stellt Beschleunigungs- und Verzögerungsrampen für die Roboter-Bewegungsbefehle Go, Jump und Pulse ein bzw. zeigt sie an.

Syntax

Accel [*accel*, *decel* [, *Zupa, Zupd, Zdna, Zdnd*]]

Parameter

accel	Integer-Ausdruck zwischen 1-100, der für einen Prozentteil der maximalen Beschleunigungsrampe steht.
decel	Integer-Ausdruck zwischen 1-100, der für einen Prozentteil der maximalen Verzögerungsrampe steht.
Zupa	Optional. Z-Achse Beschleunigung aufwärts. Gültige Einträge sind 1-100
Zupd	Optional. Z-Achse Verzögerung aufwärts. Gültige Einträge sind 1-100
Zdna	Optional. Z-Achse Beschleunigung abwärts. Gültige Einträge sind 1-100
Zdnd	Optional. Z-Achse Verzögerung abwärts. Gültige Einträge sind 1-100

Rückgabewerte

Wenn Parameter ausgelassen werden, werden die aktuellen Accel-Parameter angezeigt.

Beschreibung

Accel spezifiziert Beschleunigung und Verzögerung für alle Bewegungen des PTP-Typs. Dies bezieht auch Bewegungen ein, die durch die Roboter-Bewegungsbefehle Go, Jump und Pulse ausgelöst wurden.

Jeder durch den **Accel**-Befehl definierte Beschleunigungs- und Verzögerungsparameter kann ein Integer-Wert von 1-100 sein. Diese Zahl steht für einen Prozentwert der erlaubten maximalen Beschleunigung (oder Verzögerung).

Der Accel-Befehl kann verwendet werden, um neue Beschleunigungs- und Verzögerungswerte einzustellen, oder einfach, um die aktuellen Werte auszudrucken. Wenn der Accel-Befehl verwendet wird, um neue Beschleunigungs- und Verzögerungswerte einzustellen, so sind die ersten 2 Parameters (*accel* und *decel*) im **Accel**-Befehl erforderlich.

Die nächsten vier Parameter (*Zupa*, *Zupd*, *Zdna*, *Zdnd*) sind jedoch optional. Diese Werte sind nur für den Jump-Befehl wirksam. Die Parameter Zupa, Zupd, Zdna, und Zdnd repräsentieren Beschleunigungs- und Verzögerungswerte für die aufwärtsgerichtete Bewegung der Z-Achse am Beginn der Jump-Bewegung und für die abwärtsgerichtete Bewegung der Z-Achse am Ende der Jump-Bewegung.

- Zupa Beschleunigungsrampe während des Jump-Befehls (Z-Achse aufwärts)
- Zupd Verzögerungsrampe während des Jump-Befehls (Z- Achse aufwärts)
- Zdna Beschleunigungsrampe während des Jump-Befehls (Z- Achse abwärts)
- Zdnd Verzögerungsrampe während des Jump-Befehls (Z- Achse abwärts)

Der **Accel**-Wert wird auf die Standardwerte zurückgesetzt (niedrige Beschleunigung) wenn eine der folgenden Aktionen ausgeführt wird:

- Einschalten der Versorgungsspannung
- Software-Reset
- Befehl MOTOR ON
- Befehl SFree oder SLock
- Befehl VERINIT
- Drücken der CTRL-C –Taste, um den Roboter anzuhalten

Einschränkungen

Ausführen des Accel-Befehls im Low-Power-Modus (Power Low):

Wird **Accel** ausgeführt, wenn sich der Roboter im Low-Power-Modus befindet (Power Low), so werden die neuen Werte gespeichert, die aktuellen Werte werden jedoch auf niedrige Werte begrenzt.

Die aktuellen Beschleunigungswerte sind wirksam, wenn Power auf High eingestellt ist und der Attend-Modus ausgeschaltet ist.

Der Unterschied zwischen den Befehlen Accel und AccelS:

Es ist wichtig zu wissen, dass der **Accel-**Befehl nicht die Beschleunigungs- und Verzögerungswerte für geradlinige- und Bogenbewegungen einstellt. (D.h. für Bewegungen, die durch die Befehle Move und Arc initiiert wurden.) Der Befehl **AccelS** wird verwendet, um Beschleunigungs- und Verzögerungsrampen für geradlinige- und Bogenbewegungen einzustellen.

Verwandte Befehle

AccelS, Go, Jump, Power, Pulse, Speed, TGo

Beispiel für eine Accel-Anweisung

Das folgende Beispiel zeigt ein einfaches Bewegungsprogramm, in welchem Beschleunigung (**Accel**) und Geschwindigkeit (Speed) unter Zuhilfenahme vordefinierter Variablen eingestellt werden.

```
Function acctest
   Integer slow, accslow, decslow, fast, accfast, decfast

   slow = 20     'setzt die slow speed-Variable
   fast = 100    'setzt die high speed-Variable
   slow = 20     'setzt die slow acceleration-Variable
   slow = 20     'setzt die slow deceleration-Variable
   slow = 100     'setzt die fast acceleration-Variable
   slow = 100     'setzt die fast deceleration-Variable

   Accel accslow, decslow
   Speed slow
   Jump pick
   On gripper
   Accel accfast, decfast
   Speed fast
   Jump place
   .
   .
   .
Fend
```

<Beispiel 2>

Nehmen Sie an, der Roboter befindet sich gegenwärtig im Low-Power-Modus (Power Low) und der Anwender versucht, über das Online-Fenster den Accel-Wert auf 100 einzustellen. Weil sich der Roboter im Low-Power-Modus befindet, wird automatisch ein maximaler Beschleunigungswert von 10 eingestellt. (Das System gestattet im Low-Power-Modus keinen Beschleunigungswert >10.)

```
>Accel 100,100
>
>Accel
Low Power Mode
    100     100
    100     100
    100     100
```

<Beispiel 3>

Stellen Sie die abwärtsgerichtete Verzögerung der Z-Achse auf langsam ein, um beim Verwenden des JUMP-Befehls ein sanftes Ablegen des Teils zu ermöglichen. Dies bedeutet, dass der *Zdnd* - Parameter niedrig gewählt werden muss, wenn die **Accel**–Werte eingestellt werden.

```
>Accel 100,100,100,100,100,35

>Accel
    100     100
    100     100
    100     35
>
```

Accel-Funktion

> F

Gibt den spezifizierten Beschleunigungswert aus.

Syntax

Accel(*Parameternummer*)

Parameter

Hinweise zur Parameternummer Integer-Ausdruck, der die folgenden Werte haben kann:

1: Angegebener Wert der Beschleunigung

2: Angegebener Wert der Verzögerung

3: Angegebener Wert der Beschleunigung in der Aufwärtsbewegung der Z-Achse während der Ausführung des JUMP-Befehls

4: Angegebener Wert der Verzögerung in der Aufwärtsbewegung der Z-Achse während der Ausführung des JUMP-Befehls

5: Angegebener Wert der Beschleunigung in der Abwärtsbewegung der Z-Achse während der Ausführung des JUMP-Befehls

6: Angegebener Wert der Verzögerung in der Abwärtsbewegung der Z-Achse während der Ausführung des JUMP-Befehls

Rückgabewerte

Integer von 1-100%

Verwandte Befehle

Accel-Anweisung

Accel Funktionsbeispiel

Dieses Beispiel benutzt die **Accel**-Funktion in einem Programm:

```
Integer currAccel, currDecel

' Holt die aktuellen Werte für accel und decel
currAccel = Accel(1)
currDecel = Accel(2)
Accel 50, 50
Jump pick
'Speichert die vorangegangenen Einstellungen
Accel currAccel, currDecel
```

AccelS Anweisung

> S

Stellt die die Beschleunigungs- und Verzögerungsrampen für geradlinige und CP-Roboter-Bewegungsbefehle ein. (z.B. Move und Arc).

Syntax

AccelS [*accel* [, *decel*]]

Parameter

accel	Integer-Ausdruck zwischen 1-5000 angegeben in mm/sec^2 , um Beschleunigungs- und Verzögerungswerte für geradlinige und CP-Roboter-Bewegungsbefehle zu definieren. Wenn *decel* ausgelassen wird, so wird *accel* verwendet, um sowohl Beschleunigungs- als auch Verzögerungsrampen zu spezifizieren.
decel	Optional. Integer-Ausdruck zwischen 1-5000 mm/sec^2 , um den Verzögerungswert zu definieren.

Rückgabewerte

Zeigt Beschleunigungs- und Verzögerungswerte an, wenn sie ohne Parameter verwendet werden.

Beschreibung

AccelS spezifiziert Beschleunigung und Verzögerung für alle Typen der Interpolationsbewegung, inklusive linearer und kurvenförmiger Interpolationen. Dies schließt die Bewegungen ein, die durch die Bewegungsbefehle Move und Arc initiiert wurden.

Der **AccelS**-Wert wird auf die Standardwerte zurückgesetzt (niedrige Beschleunigung) wenn eine der folgenden Aktionen ausgeführt wird:

- Einschalten der Versorgungsspannung
- Software-Reset
- Befehl MOTOR ON
- Befehl SFree oder SLock
- Befehl VERINIT
- Drücken der CTRL-C –Taste, um den Roboter anzuhalten

Einschränkungen

Ausführen des AccelS-Befehls im Low-Power-Modus (Power Low):

Wird **AccelS** ausgeführt, wenn sich der Roboter im Low-Power-Modus befindet (Power Low), so werden die neuen Werte gespeichert, aber die aktuellen Werte werden auf niedrige Werte begrenzt.

Die aktuellen Beschleunigungswerte sind wirksam, wenn Power auf High eingestellt ist und der Attend-Modus ausgeschaltet ist.

Der Unterschied zwischen den Befehlen Accel und AccelS:

Es ist wichtig zu wissen, dass der **AccelS-**Befehl nicht die Beschleunigungs- und Verzögerungswerte für PTP-Bewegungen einstellt. (D.h. für Bewegungen, die durch die Befehle Go, Jump, und Pulse initiiert wurden.) Beschleunigungs- und Verzögerungswerte für PTP-Bewegungen werden mit dem Accel-Befehl eingestellt.

Verwandte Befehle

Accel, Arc, Power, Move, TMove, SpeedS

AccelS Beispiel

Das folgende Beispiel zeigt ein einfaches Bewegungsprogramm, in welchem die geradlinige-/CP-Beschleunigung (AccelS) und die geradlinige-/CP-Geschwindigkeit (SpeedS) unter Zuhilfenahme vordefinierter Variablen eingestellt werden.

```
Function acctest
   Integer slow, accslow, fast, accfast

   slow = 20          'setzt die slow speed-Variable
   fast = 100         'setzt die high speed-Variable
   slow = 200         'setzt die slow acceleration-Variable
   accfast = 5000     'setzt die fast acceleration-Variable
   AccelS accslow
   SpeedS slow
   Move P1
   On 1
   AccelS accfast
   SpeedS fast
   Jump P2
   .
   .
   .
Fend
```

<Beispiel 2>
Nehmen Sie an, der Roboter befindet sich gegenwärtig im Low-Power-Modus (Power Low) und der Anwender versucht, über das Online-Fenster den AccelS-Wert auf 1000 einzustellen. Weil sich der Roboter im Low-Power-Modus befindet, wird automatisch ein maximaler Beschleunigungswert von 200 eingestellt. (Das System gestattet im Low-Power-Modus keinen Beschleunigungswert >200.)

```
>AccelS 1000

>AccelS
Low Power Mode
     1000.000    1000.000
>
```

AccelS Funktion

> F

Gibt Beschleunigung- und Verzögerung für geradlinige Bewegungsbefehle aus.

Syntax

AccelS(*Parameternummer*)

Parameter

Hinweise zur Parameternummer Integer-Ausdruck, der die folgenden Werte haben kann:

1: Angegebener Wert der Beschleunigung

2: Angegebener Wert der Verzögerung

Rückgabewerte

Integer von 1-100%

Verwandte Befehle

AccelS-Anweisung

AccelS Funktionsbeispiel

```
Integer savAccelS
savAccelS = AccelS(1)
```

Agl-Funktion

F

Angle (Winkel)

Gibt den Achswinkel für die ausgewählte Rotationsachse aus, oder die Position für die ausgewählte Linearachse.

Syntax

Agl(*Achsnummer*)

Parameter

Hinweise zur Achsnummer Integer-Ausdruck, der die Achsnummer repräsentiert.

SCARA-Roboter: Bei denen (1) die T1-Achse ist; (2) die T2- Achse; (3) die Z-Achse, und (4) die U- Achse.

Kartesische Roboter: Bei denen (1) die X- Achse ist; (2) die Y- Achse; (3) die Z-Achse, und (4) die U- Achse.

Rückgabewerte

Der Achswinkel für die ausgewählte Rotationsachse oder die Position für die ausgewählten Linearachsen.

Beschreibung

Die **Agl**-Funktion wird verwendet, um den Achswinkel für die ausgewählte Rotationsachse oder die Position für die ausgewählte Linearachse zu erhalten.

Wenn die gewählte Achse eine Rotationsachse ist, gibt **Agl** den gegenwärtigen Winkel aus. Dieser wird in Grad (°) gemessen, von der Nullposition der gewählten Achse aus. Der ausgegebene Wert ist eine Real-Zahl.

Wenn die gewählte Achse eine Linearachse ist, gibt **Agl** die aktuelle Position in Millimetern (mm) aus, von der Nullposition der gewählten Achse aus gemessen. Der ausgegebene Wert ist eine Real-Zahl.

Wurde über die Arm-Anweisung ein zusätzlicher Arm ausgewählt, gibt **Agl** den Winkel (bzw. die Position) von der Null-Pulse-Position des Standardarms zum gewählten Arm aus.

Verwandte Befehle

PAgl, Pls, PPls

Agl Funktionsbeispiel

Die folgenden Beispiele werden mittels des Print-Befehls vom Online-Fenster aus getätigt.

```
> print agl(1), agl(2)
 17.234   85.355
```

AOpen Anweisung

S

Append Open (Öffnen zum Anfügen)

Öffnet eine Datei, um Daten anzufügen.

Syntax

AOpen *Dateiname* **As** #*Dateinummer*
.
.
Close #*Dateinummer*

Parameter

Hinweise zum Dateinamen Zeichenkettenausdruck, der einen gültigen Pfad und Dateinamen spezifiziert.

Hinweise zur Dateinummer Integer-Ausdruck, der Werte von 30 – 63 repräsentiert.

Beschreibung

Öffnet die angegebene Datei und identifiziert sie durch die spezifizierte Dateinummer. Diese Anweisung wird verwendet, um Daten an eine spezifizierte Datei anzuhängen. Der Befehl **Close** schließt die Datei und gibt die Dateinummer aus.

Die spezifizierte Datei muss auf Diskette vorhanden sein, anderenfalls tritt ein Fehler auf. Die spezifizierte Dateinummer identifiziert die Datei, während sie geöffnet ist und kann nicht dazu verwendet werden, auf eine andere Datei zu verweisen, bis die erste Datei geschlossen wurde. Die Dateinummer wird von den Anweisungen Print# und **Close** verwendet.

Verwandte Befehle

Close, Print #, ROpen, WOpen

Beispiel einer AOpen-Anweisung

```
Real data(200)

WOpen "TEST.VAL" As #30
For i = 0 To 100
   Print #30, data(i)
Next I
Close #30
....
....
....
AOpen "TEST.VAL" As #30
For i = 101 to 200
   Print #30, data(i)
Next i
Close #30
```

And Operator

S

Logischer Operator, der verwendet wird, um eine logische **UND**-Verknüpfung von zwei Ausdrücken herzustellen.

Syntax

Wenn *expr1* **UND** *exrp2* Dann

Ergebnis = *exrp1* **UND** *exrp2*

Parameter

exp1 Jeder gültige Ausdruck, der ein Boolesches Ergebnis ausgibt.

exp2 Jeder gültige Ausdruck, der ein Boolesches Ergebnis ausgibt.

Rückgabewerte

Die logische **UND** –Verknüpfung zweier Ausdrücke gibt ein Boolesches Ergebnis aus. (Wahr oder Falsch)

Beschreibung

Eine logische **UND** –Verknüpfung wird normalerweise verwendet, um die Ergebnisse zweier oder mehrerer Ausdrücke in ein Boolesches Ergebnis zu kombinieren. Die folgende Tabelle listet mögliche Kombinationen auf.

expr1 (Ausdruck 1)	***expr2 (Ausdruck 2)***	***Ergebnis***
WAHR	WAHR	WAHR
WAHR	FALSCH	Falsch
FALSCH	WAHR	FALSCH
FALSCH	FALSCH	FALSCH

Verwandte Befehle

LShift, Not, Or, RShift, Xor

Beispiel des UND-Operators

```
Function andtest
   Integer X, Y, Z

   Print "Enter 1 or 2"
   Input x
   Print "Enter 3 or 4"
   Input y
   Select TRUE
      Case x = 1 And y = 3
      Case x = 1 And y = 4
      Case x = 2 And y = 3
      Case x = 2 And y = 4
      Default
         Print "X or Y value entered is invalid."
         Print "Please re-enter."
   Send
   Print "X = ", x, " And Y = ", y
Fend
```

Arc Anweisung

> S

Bewegt den Arm mittels einer kreisinterpolierten Bewegung an den spezifizierten Punkt in der XY-Ebene.

Syntax

Arc *midPoint, endPoint* [**CP**] [*searchExpr*] [*!...!*]

Parameter

Hinweise zum midpoint (Mittelpunkt) Punktausdruck. Der Mittelpunkt (vorher vom Anwender geteacht), über den der Arm auf seinem Weg vom aktuellen Punkt zum zum *endPoint (Endpunkt)* hinweg verfährt.

Hinweise zum endpoint (Endpunkt) Punktausdruck. Der Endpunkt (vorher vom Anwender geteacht), zu welchem der Arm während einer Bogenbewegung verfährt. Dies ist die Endposition am Ende der Bogenbewegung.

CP Optional. Spezifiziert die Continuous Path (kontinuierlicher Weg) -Bewegung.

Hinweise zum searchExpr (Suchausdruck) Optional. Ein Till oder Find –Ausdruck.
Till | **Find**
Till Sw(*expr*) = {**Ein** | **Aus**}
Find Sw(*expr*) = {**Ein** | **Aus**}

!...! Parallelbearbeitungsanweisungen dürfen zusammen mit der Arc-Anweisung verwendet werden. Diese sind optional. (Bitte lesen Sie die Beschreibung der Parallelbearbeitungsanweisungen für detailliertere Informationen.)

Beschreibung

Arc wird verwendet, um den Arm in einer Kreisbewegung von seiner gegenwärtigen Position über den *Mittelpunkt* zum *Endpunkt* zu verfahren. Das System berechnet automatisch eine auf drei Punkten basierende Kurve (aktuelle Position, *Endpunkt*, und *Mittelpunkt*) und bewegt sich an dieser Kurve entlang, bis der als *Endpunkt* definierte Punkt erreicht wird. Die Koordinaten von *Mittel-* und *Endpunkt* müssen geteacht werden, bevor der **Arc**–Befehl ausgeführt wird. Die Koordinaten können nicht im **Arc**–Befehl selber spezifiziert werden.

Einstellen von Geschwindigkeit und Beschleunigung für die Bogenbewegung

Geschwindigkeit und Beschleunigung für die **Arc**–Anweisung werden mithilfe der Befehle SpeedS und AccelS eingestellt. SpeedS und AccelS erlauben es dem Anwender, eine Geschwindigkeit in mm/sec und eine Beschleunigung in mm/sec^2 zu spezifizieren.

Einschränkung

Der Arc-Befehl funktioniert ausschließlich auf horizontaler Ebene

Der **Arc**-Pfad ist ein tatsächlicher Bogen in der horizontalen Ebene. Der Pfad ist interpoliert und verwendet die *endPoint*–Werte als Basis für Z und U. Dreidimensionale Bögen sind daher mit dem **Arc**-Befehl nicht möglich.

Überprüfen eines Bereichs für den Arc-Befehl

Die Arc-Anweisung kann vor der **Arc**-Bewegung keine Bereichsüberprüfung der Trajektorie berechnen. Daher gilt, dass der Roboter sogar auf dem Weg zu Zielpositionen, die sich innerhalb eines erlaubten Bereichs befinden, versuchen kann, einen Pfad entlang zu fahren, der einen ungültigen Bereich hat. In diesem Fall hält der Manipulator abrupt an und kann dadurch erheblich beschädigt werden. Um dies zu vermeiden, führen Sie Bereichsüberprüfungen aus, wenn das Programm bei langsamer Geschwindigkeit operiert. Lassen Sie das Programm erst danach bei höherer Geschwindigkeit laufen.

Hinweise

Vorgeschlagene Bewegung um die Arc-Bewegung einzurichten

Da die **Arc**-Bewegung von der aktuellen Position aus beginnt, kann es notwendig sein, den Go, Jump oder einen anderen verwandten Bewegungsbefehl zu verwenden, um den Roboter in die gewünschte Position zu bringen, bevor der **Arc**-Befehl ausgeführt wird.

Potentielle Fehler

Änderung der Armattribute

Achten Sie besonders auf die Armattribute der Punkte, die mit dem **Arc**-Befehl verwendet werden. Wenn sich die Ausrichtung des Arms während der Interpolationsbewegung ändert (von rechtsarmiger zu linksarmiger Orientierung oder umgekehrt), tritt ein Fehler auf. Dies bedeutet, dass die Armattributwerte (/L Lefty, oder /R Righty) für die aktuelle Position, den *Mittelpunkt* und den *Endpunkt* identisch sein müssen.

Versuch, den Arm außerhalb des Arbeitsbereiches zu bewegen

Wenn die spezifizierte Kreisbewegung versucht, den Arm außerhalb seines Arbeitsbereiches zu bewegen, tritt ein Fehler auf.

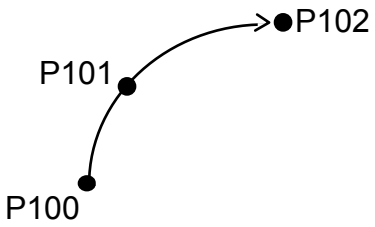
Verwandte Befehle

!Parallelbearbeitung!, AccelS, Move, SpeedS

Arc Beispiel

Das folgende Diagramm zeigt eine Bogenbewegung, die ihren Ursprung bei Punkt P100 hatte, sich dann durch Punkt P101 bewegt und bei Punkt P102 endet. Die folgende Funktion würde einen solchen Bogen generieren:

```
Function ArcTest
   Go P100
   Arc P101, P102
Fend
```



Annahme

Es wird angenommen, dass die Punkte P100, P101 und P102 vor Implementierung dieses Beispiels eingeteacht wurden.

Tipp

Wenn Sie den **Arc**-Befehl zum ersten Mal verwenden, ist es ratsam, einen einfachen Bogen mit Punkten direkt vor dem Roboter und ungefähr in der Mitte seines Arbeitsbereiches zu wählen. Versuchen Sie, sich den zu generierenden Bogen bildlich vorzustellen und stellen Sie sicher, dass Sie keine Punkte einteachen, für deren Erreichen der Roboter versuchen müsste, sich außerhalb seines normalen Arbeitsbereiches zu bewegen.

Arch Anweisung

> S

Definiert die **Arch** Parameter (Z-Hubstrecke muss zurückgelegt werden, bevor die Horizontalbewegung beginnt) oder zeigt sie für die Verwendung mit dem Jump-Befehl an.

Syntax

Arch [*archNumber, riseDist, fallDist*]

Parameter

Hinweise zur archNumber (Bogen-Nummer) Integer-Ausdruck, der die zu definierende **Arch**-Nummer repräsentiert. Gültige **Arch-**Nummern sind (0-6), was eine Gesamtanzahl von 7 Einträgen in die **Arch**-Tabelle ergibt. (Siehe auch die Standard-**Arch-**Tabelle weiter unten)

riseDist Die zurückgelegte vertikale Distanz (Hubbewegung, Z) zu Beginn der Jump-Bewegung und vor Anfang der horizontalen Bewegung. (Angegeben in Millimetern)

fallDist Die nötige vertikale Distanz (Absenkstrecke, gemessen von der Z-Position des Punktes, auf den sich der Arm zubewegt), um nach vollständigem Abschluss der Horizontalbewegung komplett vertikal zu verfahren. (Angegeben in Millimetern)

Rückgabewerte

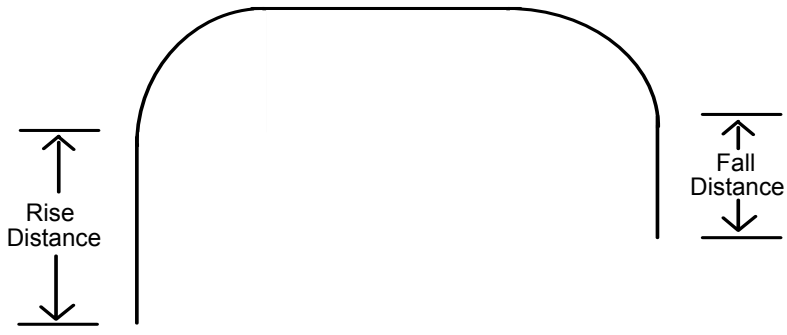
Zeigt die Arch-Tabelle an, wenn sie ohne Parameter genutzt wird.

Der gesamt Inhalt der Arch-Tabelle wird angezeigt, wenn der Arch-Befehl ohne Parameter über das Online-Fenster eingegeben wird.

Beschreibung

Der Hauptzweck des **Arch**-Befehls liegt darin, Werte in der **Arch**-Tabelle zu definieren, die für die Nutzung mit dem Jump-Bewegungsbefehl benötigt werden. Die **Arch**-Bewegung wird über die Parameter ausgeführt, welche der Arch-Nummer entsprechen, die in der Jump C Bedingung ausgewählt wurde. (Für ein vollständiges Verständnis des **Arch**-Befehls ist es notwendig, zunächst die Funktionsweise des Jump-Befehls zu begreifen.)

Die Arch-Definitionen gestatten es dem Anwender, bei Verwendung des Jump-C-Befehls in Z-Richtung “Ecken abzurunden”. Während der Jump-Befehl den Punkt angibt, auf den die Bewegung gerichtet ist (inklusive der endgültigen Position der Z-Achse), spezifizieren die Einträge in der **Arch**-Tabelle, wie lang die Hubstrecke sein muss, bevor die horizontale Bewegung (*riseDist*) einsetzt, und welche Distanz von der endgültige Position der Z-Achse zurückgelegt werden muss, um die gesamte horizontale Bewegung abzuschließen (*fallDist, Absenkstrecke*). (Siehe unten stehendes Diagramm)



Die **Arch-**Definitionstabelle enthält insgesamt 8 Einträge, von denen 7 (0-6) benutzerdefinierbar sind. Der 8. Eintrag (**Arch** 7) ist der Standard-Arch, der gar nicht wirklich einen Arch spezifiziert – ein Vorgang, der Gate-Bewegung genannt wird. (Siehe unten stehendes Gate-Bewegungs-Diagramm). Verwendet mit der Standard-**Arch**-Eingabe (Eingabe 8) veranlasst der Jump-Befehl den Arm zu folgender Bewegung:

1) Beginn der Bewegung mit ausschließlichem Verfahren der Z-Achse, bis sie den durch den LimZ-Befehl vorgegebenen Z-Koordinatenwert erreicht hat. (Oberer Z-Wert)
2) Als nächstes Bewegung in horizontaler Richtung zur Zielpunktposition, bis die endgültigen X-, Y- und U- Positionen erreicht sind.
3) Der Jump-Befehl wird dann abgeschlossen, indem der Arm nur mit einer Bewegung der Z-Achse abwärts bewegt wird, bis die Zielposition der Z-Achse erreicht ist.

Gate Motion
(JUMP with ARCH 7)

Arch-Tabellen-Standardwerte:

Arch-Nummer	Hubstrecke	Absenkstrecke
0	30	30
1	40	40
2	50	50
3	60	60
4	70	70
5	80	80
6	90	90

Hinweise

Eine andere Ursache für die Gate-Bewegung

Wenn der spezifizierte Wert für die Hub- oder Absenkstrecke größer ist, als die tatsächliche Z-Achsen-Distanz, über die sich der Roboter bewegen muss, um die Zielposition zu erreichen, tritt eine Gate-Bewegung auf. (D.h. es tritt keine Art von **Arch**-Bewegung auf.)

Zurücksetzen der Arch-Tabelle auf die Standardwerte (mithilfe des Verinit-Befehls)

Mit jeder Ausgabe des Verinit-Befehls werden die Werte der **Arch**-Tabelle auf ihre Standardwerte zurückgesetzt.

Die Arch-Werte werden beibehalten

Die Werte der **Arch-**Tabelle werden dauerhaft gespeichert und nicht geändert, bis sie entweder vom Anwender geändert werden, oder ein Verinit-Befehl ausgegeben wird.

Verwandte Befehle

Jump, Verinit

Arch-Beispiel

Im Folgenden finden Sie Beispiele von Arch-Einstellungen, die über das Online-Fenster vorgenommen wurden.

```
> arch 0, 15, 15
> arch 1, 25, 50
> jump p1 c1
> arch
  arch0 =    15.000        15.000
  arch1 =    25.000        50.000
  arch2 =    50.000        50.000
  arch3 =    60.000        60.000
  arch4 =    70.000        70.000
  arch5 =    80.000        80.000
  arch6 =    90.000        90.000
>
```

Arch Funktion

> F

Gibt die Arch-Einstellungen aus.

Syntax

Arch (*archNumber, paramNumber*)

Parameter

Hinweise zur archNumber (Bogen-Nummer) Integer-Ausdruck, welcher die Arch-Einstellungen zur Ausgabe eines Parameters (0 bis 6) repräsentiert.

Hinweise zur Parameternummer 1: vertikale Hubstrecke

2: vertikale Absenkstrecke

Rückgabewerte

Eine Real-Zahl, welche die Strecke beinhaltet.

Verwandte Befehle

Arch-Anweisung

Arch Funktionsbeispiel

```
Real archValues(6, 1)
Integer i

'Speichert die aktuellen Arch-Werte
For i = 0 to 6
    archValues(i, 0) = Arch(i, 1)
    archValues(i, 1) = Arch(i, 2)
Next i
```

Arm Anweisung

> S

Wählt die zu verwendende Armnummer aus bzw. zeigt sie an.

Syntax

Arm [*armNumber*]

Parameter

Hinweise zur Armnummer (armNumber) Optionaler Integer-Ausdruck. Gültiger Bereich: 0 - 3. Der Anwender kann bis zu vier verschiedene Arme auswählen. Arm 0 ist der Standard- (Vorgabe-) Roboterarm. Die Arme 1 - 3 sind Zusatzarme, die über die Nutzung des ArmSet-Befehls definiert werden. Werden sie ausgelassen, wird die Nummer des aktuellen Arms angezeigt.

Rückgabewerte

Wenn der **Arm-**Befehl ohne Parameter ausgeführt wird, zeigt das System die aktuelle Armnummer an.

Beschreibung

Die Arm-Anweisung gestattet es dem Anwender zu spezifizieren, welcher Arm für Roboterbefehle zu benutzen ist. Der **Arm-**Befehl gestattet es jedem Zusatzarm, normale Positionsdaten zu verwenden. Wenn keine Zusatzarme installiert sind, operiert der Standardarm (Arm Nummer 0). Da zum Auslieferungszeitpunkt werksseitig die Armnummer 0 spezifiziert ist, ist es nicht notwendig, den **Arm**-Befehl zur Auswahl eines Armes zu benutzen. Werden jedoch Zusatzarme genutzt, so müssen sie zuerst mit dem ArmSet-Befehl definiert werden.

Die Konfigurationsmöglichkeit des Zusatzarmes erlaubt es dem Anwender, die richtigen Roboterparameter für seinen Roboter zu konfigurieren, sollte die tatsächliche Roboterkonfiguration sich leicht von der eines Standardroboters unterscheiden. Wenn der Anwender beispielsweise eine zweite Ausrichtungsachse an die zweite Roboterachse montiert hat, wird der Anwender wahrscheinlich die richtigen Roboterachsverbindungen für den neu angegliederten Roboterarm definieren wollen Dies ermöglicht die korrekte Funktionsweise des neu definierten Roboterarmes unter den folgenden Bedingungen:

- ein einzelner Datenpunkt soll von zwei oder mehr Armen angefahren werden.
- bei Verwendung des Befehls PALLET
- Bei Angabe einer CP-Bewegung
- bei Spezifikation einer relativen Position
- bei Verwendung lokaler Koordinaten

Beim Betrieb eines SCARA-Roboters mit drehbaren Achsen oder Robotern mit Zylinderkoordinaten in einem kartesischen Koordinatensystem werden die Berechnungen zu den Achswinkeln auf der Basis der ArmSet-Parameter ausgeführt. Daher ist dieser Befehl kritisch, wenn eine Definition für einen Zusatzarm oder eine –Hand benötigt wird.

Hinweise

Arm 0

Arm 0 kann vom Anwender nicht über den ArmSet-Befehl definiert oder geändert werden. Er ist reserviert, da er dazu verwendet wird, die Standard-Roboterkonfiguration zu definieren. Wenn der Anwender ‘Arm’ auf 0 setzt, bedeutet dies, dass die Standardparameter des Roboterarmes verwendet werden sollen.

Armnummer nicht definiert

Die Anwahl einer zuvor nicht durch den ArmSet-Befehl definierten Zusatzarm-Nummer führt zu einem Fehler.

Verwandte Befehle

ArmSet

Beispiel einer Arm-Anweisung

Die folgenden Beispiele enthalten potentielle Zusatzarm-Definitionen, welche die Befehle ArmSet und Arm verwenden. Der ArmSet-Befehl definiert den Zusatzarm und die Arm-Anweisung definiert, welcher Arm als aktueller Arm genutzt wird. (Arm 0 ist der Vorgabe-Roboterarm und kann vom Anwender nicht eingestellt werden.)

Über das Online-Fenster:

```
> ArmSet 1, 300, -12, -30, 300, 0
> ArmSet
  arm0 250  0  0  300  0
  arm1 300  -12  -30  300  0

> Arm 0
> Jump P1     'Springt unter Verwendung von Standard Arm-Config zu P1
> Arm 1
> Jump P1     'Springt unter Verwendung des Zusatzarmes 1 zu P1
```

Arm Funktion

> F

Gibt die aktuelle Armnummer aus.

Syntax

Arm

Rückgabewerte

Integer, welcher die aktuelle Armnummer enthält.

Verwandte Befehle

Arm Anweisung

Arm Funktionsbeispiel

```
Print "The current arm number is: ", Arm
```

ArmSet Anweisung

> S

Spezifiziert Zusatzarme und zeigt sie an.

Syntax

ArmSet [*armNumber* , *link2Dist*, *joint2Offset, zOffset,* [**,***link1Dist*, *orientAngOffset*]]

Parameter

Hinweise zur Armnummer (armNumber) Integer-Ausdruck: Gültiger Bereich von 1-3. Der Anwender kann bis zu 3 verschiedene Zusatzarme definieren.

Hinweise zur link2Dist **(Für SCARA-Roboter)** Die horizontale Entfernung von der Mittellinie des Ellbogengelenks zur Mittellinie der neuen Ausrichtungsachse. (D.h. die Position an der sich die die Mittellinie der Ausrichtungsachse des neuen Zusatzarmes befindet.)

(Für kartesische Roboter) Der Positionsversatz in X-Achsen-Richtung zur ursprünglichen X-Position wird in mm angegeben.

Hinweise zum joint2Offset **(Für SCARA-Roboter)** Der Versatz (angegeben in Grad°) zwischen der Linie, die zwischen der normalen Ellbogen-Mittellinie und der normalen Mittellinie der Ausrichtungsachse gezogen wird, und der Linie, die zwischen der Ellbogen-Mittellinie des neuen Zusatzarmes und der neuen Mittellinie der Ausrichtungsachse gezogen wird. (Diese zwei Linien sollten sich an der Ellbogen-Mittellinie überschneiden und der dadurch gebildete Winkel heißt *joint2Offset*.)

(Für kartesische Roboter) Der Positionsversatz in Y-Achsen-Richtung zur ursprünglichen Y-Position wird in mm angegeben.

Hinweise zum zOffset **(Für SCARA- & kartesische Roboter)** Der Unterschied des Z-Höhenversatzes zwischen der Mitte der neuen Ausrichtungsachse und der Mitte der alten Ausrichtungsachse. (Dabei handelt es sich um eine Entfernung.)

Hinweise zur link1Dist **(Für SCARA-Roboter)** Die Entfernung der Schulter-Mittellinie zur Ellbogen-Mittellinie der Ellbogenausrichtung der neuen Zusatzachse.

(Für kartesische Roboter) Hierbei handelt es sich um einen Dummy-Parameter (Spezifiziert mit 0)

Hinweise zum orientAngOffset **(Für SCARA- & kartesische Roboter)** Der Winkelversatz (in Grad°) für die neue Ausrichtungsachse gegenüber der alten Ausrichtungsachse.

Rückgabewerte

Wenn der **ArmSet**-Befehl ohne Parameter initiiert wird, zeigt das System die aktuellen Definitionsparameter des Zusatzarms an. **ArmSet** gibt jedoch keine Rückgabewerte aus.

Beschreibung

Der Befehl gestattet es dem Anwender, Zusatzarm-Parameter zu spezifizieren, die zusätzlich zur Konfigurierung des Standard-Arms verwendet werden können. Dies ist sehr nützlich, wenn am Roboter ein Zusatzarm oder eine Zusatzhand angebracht sind. Wenn ein Zusatzarm verwendet wird, so wird er durch den Arm-Befehl ausgewählt.

Die Parameter *link1Dist* und *orientAngOffset* sind optional. Wenn sie ausgelassen werden, werden die Vorgabewerte als Standard-Armwerte verwendet.

Die Konfigurationsmöglichkeit des Zusatzarmes erlaubt es dem Anwender, die richtigen Roboterparameter für seinen
Roboter zu konfigurieren, sollte die tatsächliche Roboterkonfiguration sich leicht von der eines Standardroboters unterscheiden. Wenn der Anwender beispielsweise eine zweite Ausrichtungsachse an die zweiten Roboterachse montiert hat, wird der Anwender wahrscheinlich die richtigen Roboterachsverbindungen für den neu angegliederten Roboterarm definieren wollen.

Dies ermöglicht die korrekte Funktionsweise des neu definierten Roboterarmes unter den folgenden Bedingungen:

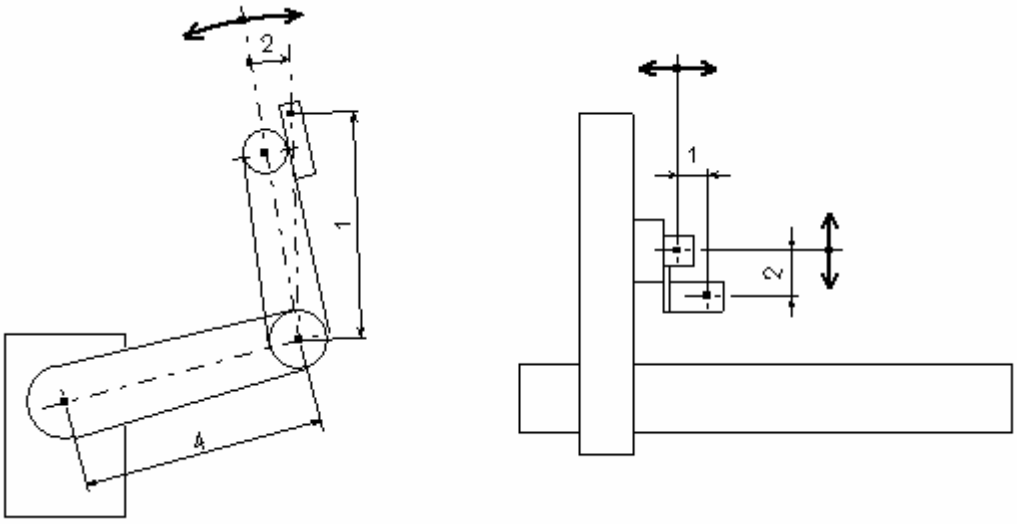
- ein einzelner Datenpunkt soll von zwei oder mehr Armen angefahren werden.
- bei Verwendung des Befehls PALLET
- bei Angabe einer CP-Bewegung
- bei Spezifikation einer relativen Position
- bei Verwendung lokaler Koordinaten

Beim Betrieb eines SCARA-Roboters mit drehbaren Achsen oder Robotern mit Zylinderkoordinaten in einem kartesischen Koordinatensystem werden die Berechnungen zu den Achswinkeln auf der Basis der **ArmSet**-Parameter ausgeführt. Daher ist dieser Befehl kritisch, wenn eine Definition für einen Zusatzarm oder eine –Hand benötigt wird.

Hinweise

Arm 0

Arm 0 kann durch den Anwender nicht definiert oder geändert werden. Er ist reserviert, da er dazu verwendet wird, die Standard-Roboterkonfiguration zu definieren. Wenn der Anwender ‘Arm’ auf 0 setzt, bedeutet dies, dass die Standardparameter des Roboterarmes verwendet werden sollen.



Verwandte Befehle

Arm

Beispiel einer ArmSet-Anweisung

Die folgenden Beispiele enthalten potentielle Zusatzarm-Definitionen, welche die Befehle **ArmSet** und Arm verwenden. Der **ArmSet**-Befehl definiert den Zusatzarm. Die Arm-Anweisung definiert, welcher Arm als aktueller Arm genutzt wird. (Arm 0 ist der Vorgabe-Roboterarm und kann vom Anwender nicht eingestellt werden.)

Über das Online-Fenster:

```
> ArmSet 1, 300, -12, -30, 300, 0
> ArmSet
  arm0 250  0  0  300  0
  arm1 300  -12  -30  300  0

> Arm 0
> Jump P1     'Springt unter Verwendung von Standard Arm-Config zu P1
> Arm 1
> Jump P1     'Springt unter Verwendung des Zusatzarmes 1 zu P1
```

ArmSet Funktion

> F

Gibt einen ArmSet Parameter aus.

Syntax

ArmSet(*armNumber, paramNumber*)

Parameter

Hinweise zur Armnummer (armNumber) Integer-Ausdruck, der die Armnummer repräsentiert, für welche Werte ausgegeben werden sollen.

Hinweise zur Parameternummer Integer-Ausdruck, der die auszugebenden Parameter (0 bis 5) repräsentiert, wie unten beschrieben.

SCARA-Roboter

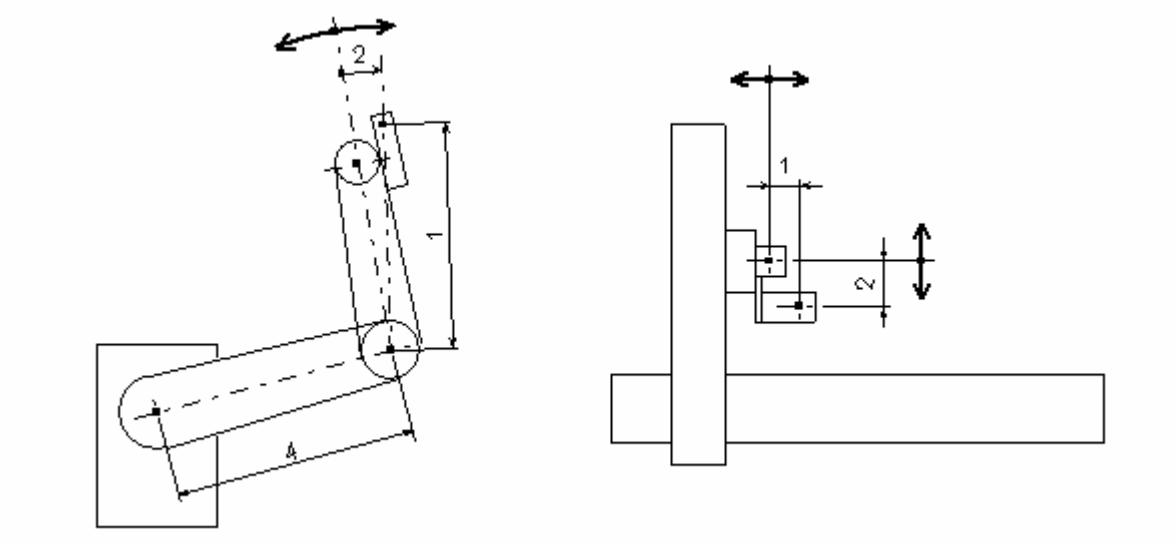
paramNumber (Parameternummer)	**Ausgegebener Wert**
1	Horizontale Entfernung von Achse #2 zur Ausrichtungsmitte (in mm)
2	Achse #2 Winkelversatz (in Grad°)
3	Höhenversatz (in mm)
4	Horizontale Entfernung von Achse #1 zu Achse #2 (in mm)
5	Winkelversatz der Ausrichtungsachse in Grad°.

KARTESISCHE Roboter:

paramNumber (Parameternummer)	**Ausgegebener Wert**
1	Positionsversatz der Achsen-Richtung für die X-Koordinate (in mm)
2	Positionsversatz der Achsen-Richtung für die Y-Koordinate (in mm)
3	Höhenversatz (in mm)
4	Reserviert. Muss Null (0) betragen.
5	Winkelversatz der Ausrichtungsachse in Grad°.

Rückgabewerte

Real-Zahl, die den Wert eines spezifizierten Parameters enthält, wie unten beschrieben.



Verwandte Befehle

Armset-Anweisung

ArmSet Funktionsbeispiel

```
Real x
x = ArmSet(1, 1)
```

Asc Funktion

F

Gibt den ASCII–Code des ersten Zeichens in einer Zeichenkette aus.

Syntax

Asc(*string*)

Parameter

Hinweise zur Zeichenkette Jeder gültige Zeichenketten-Ausdruck von mindestens einem Zeichen Länge.

Rückgabewerte

Gibt einen Integer aus, der den ASCII-Code des ersten Zeichens einer Zeichenkette repräsentiert, welche zur **ASC** -Funktion geschickt wird.

Beschreibung

Die **Asc-**Funktion wird verwendet, um ein Zeichen in seine numerischen ASCII-Darstellung zu verwandeln. Bei der zur **ASC**-Funktion gesendeten Zeichenkette kann es sich sowohl um eine konstante, als auch um eine variable Zeichenkette handeln.

Hinweis

Es wird nur der ASCII-Code des ersten Zeichens ausgegeben

Obgleich der **Asc-**Befehl Zeichenketten gestattet, die länger als ein Zeichen sind, wird nur das erste Zeichen auch durch den **Asc-**Befehl genutzt. **Asc** gibt nur den ASCII-Code des ersten Zeichens aus.

Potentielle Fehler

Verwedung einer NULL-Zeichenkette:

Wenn die **Asc**–Funktion mit einer leeren Zeichenkette oder Nullzeichenkette als Parameterwert initiiert wird, tritt ein Fehler auf.

Verwandte Befehle

Chr$, Left$, Len, Mid$, Right$, Space$, Str$, Val

Asc Funktionsbeispiel

Dieses Beispiel verwendet den **Asc**-Befehl in einem Programm und über das Online-Fenster:

```
Function asctest
    Integer a, b, c
    a = Asc("a")
    b = Asc("b")
    c = Asc("c")
    Print "The ASCII value of a is ", a
    Print "The ASCII value of b is ", b
    Print "The ASCII value of c is ", c
Fend
```

Über das Online-Fenster:

```
>print asc("a")
97
>print asc("b")
98
>
```

Atan Funktion

F

Gibt den Arkustangens eines numerischen Ausdrucks aus.

Syntax

Atan(*number***)**

Parameter

Hinweise zur number (Zahl) Numerischer Ausdruck, der den Tangens eines Winkelwertes repräsentiert.

Rückgabewerte

Real-Wert, in Radianten, der den Arkustangens des Parameters ‚*number'* repräsentiert.

Beschreibung

Atan gibt den Arkustangens des numerischen Ausdrucks aus. Der numerische Ausdruck (*number)* kann jeglicher numerische Wert sein. Der von **Atan** ausgegebenen Wert bewegt sich zwischen -PI / 2 und PI / 2. (ca. -1.57079 bis 1.5709).

Verwenden Sie die RadToDeg-Funktion, um Radiantenwerte in Gradzahlen umzuwandeln.

Verwandte Befehle

Abs, Atan2, Cos, DegToRad, Int, Mod, Not, RadToDeg, Sgn, Sin, Sqr, Str$, Tan, Val

Atan Funktionsbeispiel

Das folgende Beispiel zeigt ein einfaches Programm, welches den **Atan**-Befehl nutzt.

```
Function atantest
   Real x, y
   x = 0
   y = 1
   Print "Atan of ", x, " is ", Atan(x)
   Print "Atan of ", y, " is ", Atan(y)
Fend
```

Atan2 Funktion

F

Gibt den Winkel der imaginären Linie aus (in Radiantenwerten), welche die Punkte (0,0) und (X, Y) verbindet.

Syntax

Atan2*(X, Y)*

Parameter

X	Numerischer Ausdruck, der die X-Koordinate repräsentiert.
Y	Numerischer Ausdruck, der die Y-Koordinate repräsentiert.

Rückgabewerte

Numerischer Wert in Radianten (-PI / 2 bis +PI / 2).

Beschreibung

Atan2(*X*, *Y*) gibt den Winkel der Linie aus, welche die Punkte (0, 0) und (*X*, *Y*) verbindet. Diese trigonometrische Funktion gibt einen Arkustangenswinkel in allen vier Quadranten aus.

Verwandte Befehle

Abs, Atan, Cos, DegToRad, Int, Mod, Not, RadToDeg, Sgn, Sin, Sqr, Str$, Tan, Val

Atan2 Funktionsbeispiel

Das folgende Beispiel zeigt ein einfaches Programm, welches den **Atan2**-Befehl nutzt.

```
Function at2test
   Real x, y
   Print "Please enter a number for the X Coordinate:"
   Input x
   Print "Please enter a number for the Y Coordinate:"
   Input y
   Print "Atan2 of ", x, ", ", y, " is ", Atan2(x, y)
Fend
```

Base Anweisung

> S

Definiert das Basis-Koordinatensystem und zeigt es an.

Syntax

(1) Base (*pBase1* **:** *pRobot1*)**,** *(pBase2* **:** *pRobot2)* [, { **L** | **R** }]

(2) Base *originX, originY, originZ, originU*

Parameter

localnum	Integer von 1-15, welcher die zu definierende Nummer des lokalen Koordinatensystems repräsentiert.
pBase1, pBase2	Punktvariablen mit Punktdaten im Basiskoordinatensystem.
pRobot1, pRobot2	**Punktvariablen mit Punktdaten im absoluten Roboter-Koordinatensystem.**
L | **R**	Optional. Richte den Basisursprung an dem linken (ersten) oder rechten (zweiten) absoluten Roboterpunkt aus.
originX	Ursprung der X-Achse im Basiskoordinatensystem.
originY	Ursprung der Y-Achse im Basiskoordinatensystem.
originZ	Ursprung der Z-Achse im Basiskoordinatensystem.
originU	Rotationswinkel des Basiskoordinatensystems.

Beschreibung

Definiert das lokale Koordinatensystem, indem dessen Ursprung und der Rotationswinkel im Verhältnis zum Roboter-Koordinatensystem angegeben werden.

Um das Basiskoordinatensystem auf seine Vorgabewerte zurückzusetzen, führen Sie die folgende Anweisung aus. Dies macht das Basiskoordinatensystem dem Roboter-absoluten Koordinatensystem gleich.

```
Base 0, 0, 0, 0
```

Hinweise

Änderungen des Basiskoordinatensystems löscht alle lokalen Definitionen

Alle lokalen Koordinatensysteme sind undefiniert. Sie müssen nach der Ausführung des BASE-Befehls neu definiert werden.

Verwandte Befehle

Local

Beispiel der Base Anweisung

Definiert den Ursprung des Basiskoordinatensystems bei 100 mm auf der X-Axis und bei 100 mm auf der Y-Achse.

```
> Base 100, 100, 0, 0
```

BClr Function

F

Löscht ein Bit in einer Zahl und gibt den neuen Wert aus.

Syntax

BClr (*number, bitNum*)

Parameter

Hinweise zur number (Zahl) Spezifiziert den numerischen Wert, dessen Bit eingestellt wird, durch einen Ausdruck oder numerischen Wert.

Hinweise zur bitNum (Bitnummer) Spezifiziert das Bit (positive Integer), welches eingestellt werden soll, durch einen Ausdruck oder einen numerischen Wert.

Rückgabewerte

Gibt den neuen Wert des spezifizierten numerischen Wertes (Integer) aus.

Verwandte Befehle

BSet, BTst

BClr Beispiel

```
flags = BClr(flags, 1)
```

Beep Anweisung

S

Lässt einen Signalton durch den Lautsprecher des Rechners ertönen.

Syntax

Beep

Verwandte Befehle

Beispiel der Beep Anweisung

```
Beep
Print "Initializing robot"
```

Boolean Anweisung

S

Deklariert Boolesche Variablen (1 Byte Integer).

Syntax

Boolean *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]**...**

Parameter

Hinweise zum varName (Variablenname) Variablenname, den der Anwender als **Boolean** deklarieren will.

Hinweise zu den subscripts Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt

(*dim1* [, *dim2* [, *dim3*]]**)**

dim1, *dim2*, *dim3* können ein Integer zwischen 0-32767 sein.

Beschreibung

Boolean wird verwendet, um Variablen als **Boolean** zu deklarieren. **Boolean** Variablen können einen von zwei Werten beinhalten, 0 = Falsch und -1 = Wahr. Alle Variablen sollten ganz oben in einer Funktion deklariert sein.

Verwandte Befehle

Byte, Double, Global, Integer, Long, Real, String

Beispiel einer Boolean Anweisung

```
Boolean partOK
Boolean A(10)         'Eindimensionale Feldvariable aus Booleschen
Boolean B(10, 10)     'Zweidimensionale Feldvariable aus Booleschen
Boolean C(10, 10, 10) 'Dreidimensionale Feldvariable aus Booleschen

partOK = CheckPart()
If Not partOK Then
   Print "Part check failed"
EndIf
```

BSet Function

F

Stellt ein Bit in einer Zahl ein und gibt den neuen Wert aus.

Syntax

BSet (*number*, *bitNum*)

Parameter

Hinweise zur number (Zahl)	Spezifiziert den Wert, dessen Bit durch einen Ausdruck oder numerischen Wert einstellt wird.
Hinweise zur bitNum (Bitnummer)	Spezifiziert das Bit (positive Integer), das durch einen Ausdruck oder einen numerischen Wert eingestellt werden soll.

Rückgabewerte

Gibt den Eingestellten Bit-Wert des spezifizierten numerischen Wertes (Integer) aus.

Verwandte Befehle

BClr, BTst

BSet Beispiel

```
flags = BSet(flags, 1)
```

BTst Funktion

F

Syntax

BTst (*number*, *bitNum*)

Parameter

Hinweise zur number (Zahl)	Spezifiziert die Nummer für den Bit-Test mit einem Ausdruck oder numerischem Wert.
Hinweise zur bitNum (Bitnummer)	Spezifiziert das zu testende Bit (positiver Integer).

Rückgabewerte

Gibt die Resultate des Bit-Tests (Integer 1 oder 0) des spezifizierten numerischen Wertes aus.

Verwandte Befehle

BClr, BSet

BTst Beispiel

```
If BTst(flags, 1) Then
    Print "Bit 1 is set"
End If
```

Byte Anweisung

S

Deklariert Byte-Variablen (1 Byte Integer).

Syntax

Byte *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]

Parameter

Hinweise zum varName (Variablenname) Variablenname, den der Anwender als **Byte** deklarieren will.

Hinweise zu den subscripts Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt

(*dim1* [, *dim2* [, *dim3*]]**)**

dim1, *dim2*, *dim3* können jeweils ein Integer zwischen 0-32767 sein.

Beschreibung

Byte wird verwendet, um Variablen als **Byte** zu deklarieren. **Byte**-Variablen können Integers im Wertebereich von -128 bis +127 enthalten. Variablen sollten in einer Funktion ganz oben deklariert sein.

Verwandte Befehle

Boolean, Double, Global, Integer, Long, Real, String

Byte Beispiel

Das folgende Beispiel deklariert eine **Byte**-Variable und ordnet ihr dann einen Wert zu. Dann wird eine Bit-weise “UND”-Verknüpfung durchgeführt, um zu sehen, ob das hohe Bit des Wertes in der Variablen ‚Test_ok’ Ein-(1) oder Aus-(0) geschaltet ist. Das Ergebnis wird auf Bildschirm ausgegeben. (Natürlich ist in diesem Beispiel das hohe Bit der Variablen ‘test_ok’ immer eingestellt, da wir der Variablen den Wert 15 zugeordnet haben.)

```
Function Test
    Byte A(10)         'Eindimensionale Feldvariable aus Bytes
    Byte B(10, 10)     'Zweidimensionale Feldvariable aus Bytes
    Byte C(10, 10, 10) 'Dreidimensionale Feldvariable aus Bytes
    Byte test_ok
    test_ok = 15
    Print "Initial Value of test_ok = ", test_ok
    test_ok = (test_ok And 8)
    If test_ok <> 8 Then
       Print "test_ok high bit is ON"
    Else
       Print "test_ok high bit is OFF"
    End If
Fend
```

Calib Anweisung

> S

Ersetzt die aktuellen Pulse-Werte der Armstellung durch die aktuellen CalPls-Werte.

Syntax

Calib *jointNumber1* [,*jointNumber2* [, *jointNumber3* [,*jointNumber4*]]]

Parameter

jointNumber (Achsnummer) Integer-Zahl von 1 bis 4, welche die zu kalibrierende Achsnummer repräsentiert. Während normalerweise zu einem bestimmten Zeitpunkt nur jeweils eine Achse kalibriert werden muss, so kann man mithilfe des **Calib**-Befehls alle vier Achsen gleichzeitig kalibrieren.

Beschreibung

Berechnet und spezifiziert automatisch den Offset-Wert (Versatzwert, Hofs). Dieser Versatz ist notwendig, um den Ursprung eines jeden Roboterachsenmotors an den entsprechenden mechanischen Ursprung des Roboters anzugleichen.

Der **Calib-**Befehl sollte verwendet werden, wenn sich der Motor-Pulse-Wert geändert hat. Zumeist kommt dieser Befehl nach dem Austausch eines Motors zur Anwendung. Normalerweise passen die Pulse-Werte der Kalibrierungsposition zu den CalPls-Pulse-Werten. Nach Wartungsarbeiten, wie z.B. dem Austausch von Motoren, passen diese Werte jedoch nicht mehr zusammen, was eine Kalibrierung notwendig macht.

Die Kalibrierung kann ausgeführt werden, indem Sie den Arm in die gewünschte Kalibrierungsposition bringen und dann den **Calib**-Befehl ausführen. Durch die Ausführung des **Calib-**Befehls wird der Pulse-Wert der Kalibrierungsposition in den CalPls-Wert geändert (korrekter Pulse-Wert für die Kalibrierungsposition).

Um eine ordnungsgemäße Kalibrierung durchzuführen, müssen Hofs-Werte bestimmt werden. Um Hofs-Werte automatisch zu berechnen bewegen Sie den Arm in die gewünschte Kalibrierungsposition und führen Sie den **Calib-**Befehl aus. Die Steuerung berechnet Hofs-Werte automatisch auf Basis der Pulse-Werte der Kalibrierung und der CalPls-Pulse-Werte.

Hinweise

Vorsicht beim Verwenden des Calib-Befehls

Die Verwendung des **Calib**-Befehls ist ausschließlich für Wartungszwecke vorgesehen. Führen Sie daher **Calib** nur aus, wenn notwendig.

Die Ausführung von **Calib** bewirkt, dass der Hofs-Wert ersetzt wird. Da unbeabsichtigte Änderungen des Hofs-Wertes unvorhersehbare Roboterbewegungen verursachen können, lassen Sie bei der Ausführung des **Calib**-Befehls Vorsicht walten und verwenden Sie ihn nur, wenn notwendig.

Potentielle Fehler

Fehler: Keine Achsnummer spezifiziert

Wenn die Achsnummer nicht mit dem **Calib-**Befehl spezifiziert wurde, tritt ein Fehler auf.

Verwandte Befehle

CalPls, Hofs

Calib Beispiel

Das folgende Beispiel wird vom Monitor aus ausgeführt.

```
> CalPls   'Zeigt die aktuellen CalPls-Werte an
   65523 43320
   -1550 21351
> Pulse    'Zeigt die aktuellen Pulse-Werte an
   65526 49358
   1542  21299
> Calib 2  'Führt nur die Kalibrierung für Achse 2 aus
> Pulse    'Zeigt die (geänderten) Pulse-Werte an
   65526 43320
   -1542 21299
>
```

Call Anweisung

S

Ruft eine Anwender-Funktion auf.

Syntax

Call *funcName* [(*argList*)]

Parameter

funcName (Funktionsnamen) Name einer aufgerufenen Funktion.

argList (Argumenten-Liste) Optional. Liste von Argumenten, die in der Funktionsdeklaration spezifiziert wurden.

Beschreibung

Der Call-Befehl überträgt die Programmsteuerung an eine Funktion, (definiert in Function...Fend). Das bedeutet, dass der **Call-**Befehl die Programmausführung veranlasst, die aktuelle Funktion zu verlassen und zu der vom **Call-**Befehl spezifizierten Funktion zu wechseln. Die Programmausführung fährt dann in dieser Funktion fort, bis ein Exit Funktion oder ein Fend-Befehl erreicht wird. Bei der nächsten Anweisung nach dem **Call**-Befehl wird die Steuerung dann wieder an die ursprüngliche Aufruf-Funktion zurückgegeben.

Sie können die Klammern für Call-Schlüsselwort und Argument weglassen. Als Beispiel sehen Sie hier eine Call-Anweisung, die mit oder ohne Call-Schlüsselwort verwendet wird:

```
Call MyFunc(1, 2)
MyFunc 1, 2
```

Verwenden Sie GoSub...Return um ein Unterprogramm innerhalb einer Funktion auszuführen.

Verwandte Befehle

Function, GoSub

Beispiel einer Call Anweisung

<File1: MAIN.PRG>

```
Function main
    Call InitRobot(1)
Fend
```

<File2: INIT.PRG>

```
Function InitRobot(robotNumber As Integer)

    Integer savRobot

    savRobot = Robot
    Robot robotNumber
    Motor On
    If Not MCalComplete Then
        MCal
    End If
    Robot savRobot
Fend
```

CalPls Anweisung

> S

Spezifiziert Position und Orientierungs-Pulse-Werte für die Kalibrierung und zeigt sie an.

Syntax

CalPls [*j1Pulses, j2Pulses, j3Pulses, j4Pulses*]

Parameter

j1Pulses Pulse-Wert für die erste Achse. Dieses ist ein Long Integer-Ausdruck.

j2Pulses Pulse-Wert für die zweite Achse. Dieses ist ein Long Integer-Ausdruck.

j3Pulses Pulse-Wert für die dritte Achse. Dieses ist ein Long Integer-Ausdruck.

j4Pulses Pulse-Wert für die vierte Achse. Dieses ist ein Long Integer-Ausdruck.

Rückgabewerte

Werden die Parameter weggelassen, werden die aktuellen **CalPls**-Werte angezeigt.

Beschreibung

Spezifiziert den/die korrekten Positions-Pulse-Wert(e) und behält ihn/sie bei.

CalPls soll für Wartungszwecke genutzt werden, wie z.B. nach dem Austausch von Motoren oder wenn die Nullposition des Motors mit der mechanischen Nullposition des entsprechenden Armes abgeglichen werden muss. Dieses Abgleichen der Nullposition des Motors mit der mechanischen Nullposition des entsprechenden Armes wird Kalibrierung genannt.

Normalerweise passen die Pulse-Werte der Kalibrierungsposition zu den **CalPls**-Pulse-Werten. Nach der Ausführung von Wartungsarbeiten, wie z.B. dem Austausch von Motoren, passen diese Werte jedoch nicht mehr zusammen, was eine Kalibrierung notwendig macht.

Die Kalibrierung kann ausgeführt werden, indem Sie den Arm in eine bestimmte Kalibrierungsposition bringen und dann den Calib-Befehl ausführen. Durch die Ausführung des **Calib-**Befehls wird der Pulse-Wert der Kalibrierungsposition in den CalPls-Wert umgeändert (korrekter Pulse-Wert für die Kalibrierungsposition).

Um eine ordnungsgemäße Kalibrierung durchzuführen, müssen Hofs-Werte bestimmt werden. Um Hofs-Werte automatisch zu berechnen, bewegen Sie den Arm in die gewünschte Kalibrierungsposition und führen Sie den Calib**-**Befehl aus. Die Steuerung berechnet Hofs-Werte automatisch auf Basis der Positions-Pulse-Werte der Kalibrierung und der **CalPls**-Werte.

Hinweise

CalPls Werte können nicht durch Ein- und Ausschalten der Versorgungsspannung geändert werden

CalPls-Werte werden nicht initialisiert, indem man die Versorgungsspannung zur Steuerung aus- und danach wie einschaltet. Die einzige Methode, die **CalPls-Werte** zu ändern, besteht in der Ausführung des **CalPls**-Befehls.

Verwandte Befehle

Calib, Hofs

Beispiel einer CalPls Anweisung

Das folgende Beispiel wird vom Online-Fenster aus ausgeführt.

```
> CalPls   'Zeigt die aktuellen CalPls-Werte an
   65523 43320
   -1550 21351
> Pulse
   65526 49358
   -1542 21299
> Calib 4
> Pulse
   65526 49358
   -1542 21351
>
```

CalPls Funktion

F

Gibt den Pulse-Wert der Kalibrierung aus, der durch die CalPls-Anweisung spezifiziert wird.

Syntax

CalPls(*joint*)

Parameter

joint Integer-Ausdruck, der eine Roboter-Achsennummer oder 0 repräsentiert, um den CalPls Status auszugeben.

Rückgabewerte

Integer Wert, der die Anzahl der Kalibrierungspulse enthält. Wenn *joint* den Wert 0 hat, wird 1 oder 0 ausgegeben, je nachdem, ob CalPls ausgeführt wurde, oder nicht.

Verwandte Befehle

CalPls Anweisung

Beispiel der CalPls Funktion

Dieses Beispiel benutzt die **CalPls** -Funktion in einem Programm:

```
Function DisplayCalPlsValues
   Integer i

   Print "CalPls Values:"
   For i = 1 To 4
      Print "Joint ", i, " CalPls = ", CalPls(i)
   Next i
Fend
```

Chain Anweisung

S

Stoppt die Ausführung der aktuellen Programmgruppe und beginnt mit der Ausführung der spezifizierten Programmgruppe.

Syntax

Chain *groupName*

Parameter

groupName	Ein Zeichenkettenausdruck, der den Namen der auszuführenden Programmgruppe repräsentiert. Die Programmgruppe muss Teil des aktuellen Projektes sein.

Beschreibung

Die **Chain**-Anweisung stoppt die Ausführung des aktuellen Programms in der aktuellen Programmgruppe. Dann beginnt sie mit der Ausführung des ersten Programms in der durch die **Chain**-Anweisung spezifizierten Programmgruppe. (Das erste Programm in der Gruppe ist dasjenige, welches für die Programmgruppe als Hauptprogramm definiert wurde.)

Bitte beachten Sie, dass der Anwender die Option hat, separate Punktdateien für jede Programmgruppe zu spezifizieren. Dies bedeutet, dass wenn der **Chain**-Befehl veranlasst, eine neue Programmgruppe zu verwenden, auch eine neue Punktdatei benutzt werden kann (je nachdem, ob der Anwender separate Punktdateien für jede Programmgruppe spezifiziert hat, oder nicht).

Die **Chain**-Anweisung ist nützlich für die Ausführung der Verzweigung von einer Programmgruppe zur anderen. Der Anwender kann beispielsweise zehn unterschiedliche Produkttypen haben, von denen jedes sein eigenes Programm hat und möglicherweise seinen eigenen Satz an Teachpunkten. In diesem Falle kann die **Chain**-Anweisung verwendet werden, um die Verzweigung zum richtigen Programm herzustellen, je nachdem, um welchen Produkttyp es sich handelt.

Die **Chain**-Anweisung kann ebenfalls verwendet werden, um die aktuelle Programmgruppe erneut zu starten. Dies ist nützlich, wenn ein kritischer Fehler aufgetreten ist und Sie das Programm noch einmal von Anfang an starten wollen. Sie können die **Chain**-Anweisung auch aus einem Fehlerbehandlungsprogramm ausführen, um dies zu erreichen.

Einschränkungen

E/A-Bedingungen werden beibehalten

Da beim Ausführen der **Chain**-Anweisung kein Software-Reset durchgeführt wird, werden die Bedingungen für Ausgang und Merker beibehalten.

OnErr Ausnahme

Wenn der Anwender die **Chain**-Anweisung mit einem nicht vorhandenen *groupName* (Gruppennamen) ausführt, ruft dies einen Fehler hervor. In diesem Fall kann der OnErr-Befehl nicht zur Fehlerbehandlung verwendet werden.

Verwandte Befehle

Xqt

Beispiel der Chain Funktion

Verwenden Sie die Chain-Anweisung in Ihren Programmen, um eine andere Programmgruppe im Projekt zu starten. Wenn die **Chain**-Anweisung ausgeführt wird, lädt diese die Punkte für das Programm, das angehängt wird, und startet dann das Hauptprogramm der Gruppe.

Um die Chain-Anweisung zu nutzen, müssen die Programmgruppen aktiviert werden. Diese Aktivierung erfolgt im Projekteditor (Projekt => Edit).

Hier ist ein einfaches Beispiel, das die Chain-Anweisung verwendet.

Programmgruppe: MAINGRP
Programmname: MAIN.PRG

```
Function Main

mainLoop:
   Print "Main Menu"
   Print "1) Run widget 1"
   Print "2) Run widget 2"
   Print "3) Quit
   Input choice
   Select choice
      Case 1
         Chain "widget1"
      Case 2
         Chain "widget2"
      Case 3
         End
      Default
         GoTo mainLoop
   Send
Fend
```

Programmgruppe: WIDGET1
Programmname: WID1.PRG

```
Function Widget1
   Print "Executing widget 1"
   Chain "MAINGRP"       ' Startet die Hauptgruppe wieder
Fend
```

Programmgruppe: WIDGET2
Programmname: WID2.PRG

```
Function Widget2
   Print "Executing widget 2"
   ' Startet die Hauptgruppe wieder
Fend
```

ChDir Befehl

>

Online-Fenster-Abkürzungs. **- Cd**

Ändert das aktuelle Verzeichnis und zeigt es an.

Syntax

(1) ChDir *pathName*

(2) ChDir

Parameter

pathName	Zeichenkettenausdruck, der den Namen des neuen Standardpfades repräsentiert.

Beschreibung

(1) Ändert das spezifizierte Verzeichnis in das Standardverzeichnis.

(2) Zeigt das aktuelle Verzeichnis an. Wenn der Pfadname weggelassen wird, wird das aktuelle Verzeichnis angezeigt. Wird verwendet, um das aktuelle Verzeichnis anzuzeigen, so dies nicht bekannt ist.

Beim Einschalten wird das Stammverzeichnis zum aktuellen Verzeichnis.

Verwandte Befehle

ChDrive, Dir

ChDir Beispiel

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt.

```
> ChDir \         'Wechselt vom aktuellen Verzeichnis in das
                   Hauptverzeichnis
> ChDir..         'Wechselt vom aktuellen Verzeichnis in das übergeordete
                   Verzeichnis

> Cd \TEST\H55    'Wechselt vom aktuellen Verzeichnis nach \H55 im
                   Verzeichnis \TEST

> Cd              'Zeigt das aktuelle Verzeichnis an
A:\TEST\H55\
```

ChDrive Anweisung

S

Ändert das Standardlaufwerk für Dateioperationen.

Syntax

ChDrive *drive*

Parameter

drive Zeichenkettenausdruck oder Buchstabensymbol, der/die einen gültigen Laufwerkbuchstaben enthält.

Verwandte Befehle

ChDir, CurDrive$

Beispiel einer ChDrive Anweisung

```
ChDrive "d"
```

ChkCom Funktion

F

Gibt die Anzahl von Zeichen im Empfangspuffer einer Kommunikationsschnittstelle aus.

Syntax

ChkCom (*portNumber)*

Parameter

portNumber Integer-Ausdruck für die zu überprüfende Schnittstellennummer.

Rückgabewerte

Anzahl der empfangenen Zeichen (Integer).

Wenn die Schnittstelle keine Zeichen empfangen kann, werden die folgenden, negativen Zeichen ausgegeben, um den Schnittstellenstatus auszugeben:

-2 Schnittstelle wird von einem anderen Task verwendet
-3 Schnittstelle ist nicht offen

Verwandte Befehle

CloseCom, OpenCom, Read, Write

ChkCom Beispiel

```
Integer numChars

numChars = ChkCom(1)
```

ChkNet Funktion

F

Gibt die Anzahl von Zeichen im Empfangspuffer eines Netzwerkports aus.

Syntax

ChkNet (*portNumber)*

Parameter

portNumber Integer-Ausdruck für die zu überprüfende Schnittstellennummer.

Rückgabewerte

Anzahl der empfangenen Zeichen (Integer).

Wenn die Schnittstelle keine Zeichen empfangen kann, werden die folgenden, negativen Zeichen ausgegeben, um den Schnittstellenstatus auszugeben:

-1 Der Port ist offen, aber es wurde keine Kommunikation hergestellt
-2 Schnittstelle wird von einem anderen Task verwendet
-3 Der Port ist nicht offen

Verwandte Befehle

CloseNet, OpenNet, Read, Write

ChkNet Beispiel

```
Integer numChars

numChars = ChkNet(128)
```

Chr$ Funktion

F

Gibt das Zeichens eines numerischen ASCII-Wertes aus.

Syntax

Chr$*(number)*

Parameter

number Integer-Ausdruck zwischen 1 und 255.

Rückgabewerte

Gibt ein Zeichen aus, welches dem spezifizierten ASCII-Code entspricht, spezifiziert durch den *number*-Wert.

Beschreibung

Chr$ gibt eine Zeichenkette aus (1 Zeichen), die den ASCII-Wert des Parameters *number* hat. Wenn die spezifizierte *number* (Nummer) außerhalb des Bereiches von 1 bis 255 liegt, tritt ein Fehler auf.

Verwandte Befehle

Asc, Left$, Len, Mid$, Right$, Space$, Str$, Val

Beispiel einer Chr$ Funktion

Das folgende Beispiel deklariert eine Zeichenketten- („String“-) Variable und ordnet ihr dann die Zeichenkette „ABC“ zu. Der Chr$-Befehl wird verwendet, um die numerischen ASCII –Werte in die Zeichen "A", "B" und "C" zu konvertieren. Das &H bedeutet, dass die folgende Nummer hexadezimal repräsentiert wird (&H41 bedeutet Hex 41).

```
Function Test
    String temp$
    temp$ = Chr$(&H41) + Chr$(&H42) + Chr$(&H43)
    Print "The value of temp = ", temp$
Fend
```

Clear

S

Initialisiert den Positionsdatenbereich für den aktuellen Roboter.

Syntax

Clear

Beschreibung

Clear initialisiert den Positionsdatenbereich für den aktuellen Roboter. Dieser Befehl wird zum Löschen von im Speicher befindlichen Punktdefinitionen verwendet, bevor neue Punkte eingeteacht werden.

Verwandte Befehle

New, Plist

Beispiel einer Clear-Anweisung

Das folgende Beispiel zeigt einfache Beispiele der Verwendung des **Clear**-Befehls (vom Online-Fenster). Beachten Sie, dass beim Initiieren des Plist-Befehls keine Teach-Punkte angezeigt werden, sobald der **Clear**-Befehl erfolgt ist.

```
>P1=100,200,-20,0/R
>P2=0,300,0,20/L
>Plist
P1=100,200,-20,0/R
P2=0,300,0,20/L
>clear
>Plist
>
```

Close Anweisung

S

Die Close-Anweisung schließt eine Datei, die mit den Befehlen AOpen, ROpen, oder WOpen geöffnet wurde.

Syntax

Close #*fileNumber*

Parameter

fileNumber Integer-Ausdruck, dessen Wert zwischen 30 – 63 liegt.

Beschreibung

Schließt die durch den Dateiport *fileNumber* referenzierte Datei und gibt sie aus.

Verwandte Befehle

AOpen, Input #, Print #, ROpen, WOpen

Close Beispiel

Dieses Beispiel öffnet eine Datei, schreibt Daten in diese Datei, öffnet dieselbe Datei später wieder und liest deren Daten in eine Feldvariable.

```
Real data(200)
WOpen "TEST.VAL" As #30
For I = 0 To 100
    Print #30, data(i)
Next I
Close #30

ROpen "TEST.VAL" As #30
For I = 0 to 100
    Input #30, data(I)
Next I
Close #30
```

CloseCom Anweisung

S

Schließt die zuvor mit dem Befehl OpenCom geöffnete Kommunikationsschnittstelle.

Syntax

CloseCom #*portNum* | **All**

Parameter

portNum Integer-Ausdruck für die zu schließende Schnittstellennummer.

Verwandte Befehle

ChkCom, OpenCom

Beispiel einer CloseCom Anweisung

```
CloseCom #1
```

CloseNet Anweisung

S

Schließt den zuvor mit dem Befehl OpenNet geöffneten Netzwerkport.

Syntax

CloseNet #*portNum*

Parameter

portNum Integer-Ausdruck für die zu schließende Schnittstellennummer. Der Bereich liegt zwischen 128 und 131.

Verwandte Befehle

ChkNet, OpenNet

Beispiel einer CloseNet Anweisung

```
CloseNet #1
```

ClrScr Anweisung

S

Löscht den **EPSON RC+** Run-, den Operator- oder den Online-Fenster-Textbereich.

Syntax

ClrScr

Beschreibung

ClrScr löscht entweder den EPSON RC+ Run-Fenstertextbereich, oder den des Operators, je nachdem von wo das Programm gestartet wurde.

Wird die **ClrScr**-Anweisung aus einem Programm ausgeführt, welches vom Online-Fenster aus gestartet wurde, so wird sie ignoriert.

Verwandte Befehle

Cls

ClrScr Beispiel

Wenn dieses Beispiel vom Run- oder vom Operator-Fenster aus gestartet wird, so wird der Textbereich des Fensters gelöscht, wenn **ClrScr** ausgeführt wird.

```
Function main
    Integer i

    Do
        For i = 1 To 10
            Print i
        Next i
        Wait 3
        ClrScr
    Loop
Fend
```

Cnv_AbortTrack Anweisung

S

Bricht einen Bewegungsbefehl an den Punkt eines Conveyor-Queues ab.

Syntax

Cnv_AbortTrack [*stopZheight*]

Parameter

stopZheight Optional. Real-Ausdruck, der die Z-Position spezifiziert, zu der sich der Roboter nach Abbruch der Verfolgung bewegen sollte.

Beschreibung

Wenn ein Bewegungsbefehl zu einem Punkt eines Conveyor-Queues abläuft, kann der Befehl **Cnv_AbortTrack** ausgeführt werden, um den Bewegungsbefehl abzubrechen.

Wird *stopZHeight* weggelassen, wird der Roboter bis zum Halt verzögert, abhängig von der aktuellen QP-Einstellung.

Wird *stopZHeight* spezifiziert, wird sich der Roboter nur bis zu diesem Wert bewegen, wenn sich die Position der Z-Achse zum Zeitpunkt des Abbruchs unterhalb von *stopZHeight* befindet.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_RobotConveyor Anweisung

Beispiel der Cnv_AbortTrack Anweisung

```
' Task zur Überwachung des Roboters, dessen nachgesteuertes Bauteil in
' Förderrichtung gefahren ist
Function WatchDownstream

  Do
    If Cnv_QueLen(1, CNV_QUELEN_DOWNSTREAM) > 0 Then
   ' Bricht das Tracking auf Conveyor 1 ab und verfährt die Z-Achse des Roboters nach 0.
      Cnv_AbortTrack 1, 0
    EndIf
    Wait .1
  Loop
Fend
```

Cnv_Fine Funktion

F

Gibt die aktuelle Cnv_Fine Einstellung aus.

Syntax

Cnv_Fine(*conveyorNumber*)

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

Rückgabewerte

Real- Wert von Cnv_Fine in Millimetern.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_Fine Anweisung

Beispiel einer Cnv_Fine Funktion

```
Real f
f = Cnv_Fine(1)
```

Cnv_Fine Anweisung

S

Stellt den Wert von Cnv_Fine für einen Conveyor ein.

Syntax

Cnv_Fine *conveyorNumber* [, *fineValue*]

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
fineValue	Optional. Real-Ausdruck, der die Entfernung spezifiziert, über die das Tracking abgeschlossen wird, in Millimetern. Wird er weggelassen, wird die aktuelle Cnv_Fine –Einstellung angezeigt.

Beschreibung

Der Vorgabewert von 20mm wird automatisch eingestellt, wenn die folgenden Bedingungen eintreten:
Ein Conveyor wird erzeugt.
Die SPEL-Treiber werden gestartet.
Reset, Motor On, SFree, oder SLock werden ausgeführt.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_Fine Funktion

Beispiel einer Cnv_Fine Anweisung

```
Cnv_Fine 1, 40
```

Cnv_Name$ Funktion

F

Gibt den Namen des spezifizierten Conveyors aus.

Syntax

Cnv_Name$ (*conveyorNumber*)

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

Rückgabewerte

Eine Zeichenkette (String), die den Namen des Conveyors enthält.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_Number

Beispiel einer Cnv_Name$ Funktion

```
Print "Conveyor 1 Name: ", Cnv_Name$(1)
```

Cnv_Number Funktion

F

Gibt die Nummer eines namentlich spezifizierten Conveyors aus.

Syntax

Cnv_Number (*conveyorName*)

Parameter

conveyorName Zeichenkettenausdruck, welcher den Conveyornamen repräsentiert.

Rückgabewerte

Integer Conveyornummer.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_Name$

Beispiel einer Cnv_Number Funktion

```
Integer cnvNum

cnvNum = Cnv_Number("Main Conveyor")
```

Cnv_Point Funktion

F

Gibt einen Roboterpunkt im Koordinatensystem des spezifizierten Conveyors aus, der von Sensorkoordinaten abgeleitet wird.

Syntax

Cnv_Point (*conveyorNumber, sensorX*, *sensorY*)

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
sensorX	Real-Ausdruck für die X-Koordinate des Sensors.
sensorY	Real-Ausdruck für die Y-Koordinate des Sensors.

Rückgabewerte

Roboterpunkt im Koordinatensystem des Conveyors.

Beschreibung

Die Cnv_Point-Funktion muss verwendet werden, um Punkte zu erzeugen, die einem Conveyor-Queue hinzugefügt werden.

Bei Vision Conveyorn sind *sensorX* und *sensorY* Bild-Koordinaten der Kamera.

Bei Sensor Conveyorn können *sensorX* und *sensorY* 0 sein, da dies der Ursprung des Conveyor-Koordinatensystems ist.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_Speed

Beispiel einer Cnv_Point Funktion

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
  VGet FindParts.Part.CameraXYU(i), found, x, y, u
  Cnv_QueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_PosErr Funktion

F

Gibt die Abweichung von der aktuellen Tracking-Position im Vergleich zum Tracking-Ziel aus.

Syntax

Cnv_PosErr (*conveyorNumber*)

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

Rückgabewerte

Real- Wert in Millimetern.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_MakePoint

Beispiel einer Cnv_PosErr Funktion

```
Print "Conveyor 1 position error: ", Cnv_PosErr(1)
```

Cnv_Pulse Funktion

F

Gibt die aktuelle Position eines Conveyors in Pulsen aus.

Syntax

Cnv_Pulse (*conveyorNumber*)

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

Rückgabewerte

Long-Wert aktueller Pulse für den angegebenen Conveyor.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_Trigger

Beispiel einer Cnv_Pulse Funktion

```
Print "Current conveyor position: ", Cnv_Pulse(1)
```

Cnv_QueAdd Anweisung

S

Fügt einem Conveyor-Queue einen Roboterpunkt hinzu.

Syntax

Cnv_QueAdd *conveyorNumber*, *pointData* [, *userData*]

Parameter

conveyorNumber	Integer-Ausdruck, der die Nummer des zu verwendenden Conveyors angibt.
pointData	Der Roboterpunkt, der zum Conveyor-Queue hinzugefügt wird.
userData	Optional. Real-Ausdruck, der verwendet wird, um Anwenderdaten zusammen mit dem Punkt abzuspeichern.

Beschreibung

pointData wird zum Ende des spezifizierten Conveyor-Queues hinzugefügt. Es wird zusammen mit der aktuell gehaltenen Conveyor Pulse-Position registriert.

Wenn sich die Entfernung zwischen *pointData* und dem vorangegangenen Punkt in dem Queue unterhalb des durch Cnv_QueReject definierten Wertes befindet, werden die Punktdaten dem Queue nicht hinzugefügt und es tritt auch kein Fehler auf.

Ein Fehler tritt dann auf, wenn der Queue voll ist.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_RobotConveyor Anweisung

Beispiel einer Cnv_QueAdd Anweisung

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
  VGet FindParts.Part.CameraXYU(i), found, x, y, u
  Cnv_QueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_QueGet Funktion

F

Gibt einen Punkt aus dem spezifizierten Conveyor-Queue aus.

Syntax

Cnv_QueGet(*conveyorNumber* [, *index*])

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
index	Optional. Integer Ausdruck, welcher den Index der Daten repräsentiert, die aus dem Queue ausgegeben werden.

Rückgabewerte

Ein Roboterpunkt im Koordinatensystem des spezifizierten Conveyors.

Beschreibung

Cnv_QueGet wird verwendet, um Punkte aus dem Queue des Conveyors auszugeben. Wenn *queNumber* weggelassen wird, wird der erste Punkt im Queue ausgegeben. Anderenfalls wird ein Punkt von der angegebenen *queNumber* ausgegeben.

Cnv_QueGet löscht den Punkt nicht aus dem Queue. Sie müssen den Befehl Cnv_QueRemove verwenden, um ihn wirklich zu löschen.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueLen, Cnv_QueRemove

Beispiel einer Cnv_QueGet Funktion

```
' Springt zum ersten Teil in der Queue und verfolgt sie
Jump Cnv_QueGet(1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1
Cnv_QueRemove 1
```

Cnv_QueLen Funktion

F

Gibt die Anzahl der sich im Queue des Conveyors befindlichen Objekte aus.

Syntax

Cnv_QueLen(*conveyorNumber* [, *paramNumber*])

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

paramNumber (Parameternummer) Optional. Integer-Ausdruck, für welche Daten die Länge ausgegeben werden soll.

Symbolische Konstante	Wert	Bedeutung
CNV_QUELEN_ALL	0	Gibt die vollständige Anzahl Objekte aus, die sich in dem Queue des Conveyors befinden.
CNV_QUELEN_UPSTREAM	1	Gibt die Anzahl der Objekte entgegen der Förderrichtung aus.
CNV_QUELEN_PICKUPAREA	2	Gibt die Anzahl der Objekte im Aufnahmebereich aus.
CNV_QUELEN_DOWNSTREAM	3	Gibt die Anzahl der Objekte in Förderrichtung aus.

Rückgabewerte

Anzahl der Einträge als Integer.

Beschreibung

Man verwendet Cnv_QueLen, um herauszufinden, wie viele Objekte in einem Queue zur Verfügung stehen. Normalerweise wird man herausfinden wollen, wieviele Objekte sich im Aufnahmebereich befinden.

Sie können Cnv_QueLen auch als Argument zur Wait-Anweisung verwenden.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueGet

Beispiel einer Cnv_QueLen Funktion

```
Do
  Do While Cnv_QueLen(1, CNV_QUELEN_DOWNSTREAM) > 0
    Cnv_QueRemove 1, 0
  Loop
  If Cnv_QueLen(1, CNV_QUELEN_PICKUPAREA) > 0 Then
    Jump Cnv_QueGet(1, 0) C0
    On gripper
    Wait .1
    Cnv_QueRemove 1, 0
    Jump place
    Off gripper
    Jump idlePos
  EndIf
Loop
```

Cnv_QueList Anweisung

S

Zeigt eine Liste der Objekte an, die sich im Queue des Conveyors befinden.

Syntax

Cnv_QueList *conveyorNumber* [, *numOfItems*]

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
numOfItems	Optional. Integer-Ausdruck, der spezifiziert, wie viele Objekte angezeigt werden sollen. Wird dieser Parameter weggelassen, werden alle Objekte angezeigt.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueGet

Beispiel einer Cnv_QueList Anweisung

```
Cnv_QueList 1
```

Cnv_QueMove Anweisung

S

Bewegt Daten vom Conveyor-Queue entgegen Förderrichtung zu einem Conveyor-Queue in Förderrichtung.

Syntax

Cnv_QueMove *conveyorNumber* [, *index* [, *userData*]]

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
index	Optional. Integer-Ausdruck, welcher den Index des ersten zu bewegenden Objekts in der Queue spezifiziert.
userData	Optional. Real-Ausdruck, der verwendet wird, um Anwenderdaten zusammen mit dem Objekt abzuspeichern.

Beschreibung

Der Befehl Cnv_QueMove wird verwendet, um ein oder mehrere Objekte von einem Conveyor-Queue zu seinem zugehörigen Conveyor-Queue in Förderrichtung zu bewegen. Wenn *index* angegeben ist, werden alle Objekte vom *index* bis zum Ende des Queue verschoben.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueGet

Beispiel einer Cnv_QueMove Anweisung

```
Cnv_QueMove 1
```

Cnv_QueReject Anweisung

S

Stellt die Entfernung für die Verwerfung aus dem Queue ein und zeigt sie an.

Syntax

Cnv_QueReject *conveyorNumber* [, *rejectDistance*]

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
rejectDistance	Optional. Real-Ausdruck, der die minimal erlaubte Entfernung zwischen Teilen innerhalb eines Queue in Millimetern definiert. Wird er weggelassen, wird die aktuelle *rejectDistance*–Einstellung angezeigt.

Beschreibung

Der Befehl **Cnv_QueReject** wird verwendet, um die minimale Entfernung zwischen Teilen zu bestimmen, um doppelte Registrierung in der Queue zu vermeiden. Wenn die Teile durch das Vision-System gescannt werden, könnten sie mehr als einmal gefunden werden, sollten jedoch nur einmal registriert werden. **Cnv_QueReject** hilft dem System, Doppelregistrierungen herauszufiltern.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueReject Funktion

Beispiel einer Cnv_QueReject Anweisung

```
Cnv_QueReject 1, 20
```

Cnv_QueReject Funktion

F

Gibt die aktuelle Entfernung für die Verwerfung eines Objekts für einen Conveyor aus.

Syntax

Cnv_QueReject (*conveyorNumber*)

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

Rückgabewerte

Real-Wert in Millimetern.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueReject Anweisung

Beispiel einer Cnv_QueReject Funktion

```
Real rejectDist

RejectDist = Cnv_QueReject(1)
```

Cnv_QueRemove Anweisung

S

Entfernt Objekte aus einem Conveyor-Queue.

Syntax

Cnv_QueRemove *conveyorNumber* [, *index*]

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
index	Optional. Integer-Ausdruck, der den Index des ersten zu entfernenden Objekt definiert.

Beschreibung

Der Befehl Cnv_QueRemove wird verwendet, um einen oder mehrere Objekte aus einem Conveyor-Queue zu entfernen. Normalerweise werden Objekte aus dem Queue entfernt, wenn Sie die Bearbeitung der Daten abgeschlossen haben.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueAdd Anweisung

Beispiel einer Cnv_QueRemove Anweisung

```
Jump Cnv_QueGet(1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1

' Entfernt die Daten von dem Conveyor
Cnv_QueRemove 1
```

Cnv_QueUserData Anweisung

S

Stellt zum Queue-Eintrag gehörige Anwender-Daten ein und zeigt sie an.

Syntax

Cnv_QueUserData *conveyorNumber* [, *index* [, *userData*]]

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
index	Optional. Integer-Ausdruck, welcher den Index der Objektnummer im Queue definiert.
userData	Optional. Real-Ausdruck, der Anwender-Daten definiert.

Beschreibung

Cnv_QueUserData wird verwendet, um Ihre eigenen Daten mit jedem Objekt in einem Conveyor-Queue zu speichern. Anwender-Daten sind optional. Sie sind für den Normalbetrieb nicht vonnöten.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueUserData Funktion

Beispiel einer Cnv_QueUserData Anweisung

```
Cnv_QueUserData 1, 1, angle
```

Cnv_QueUserData Funktion F

Gibt den Wert der Anwender-Daten wieder, der mit einem Objekt in dem Conveyor-Queue verbunden ist, aus.

Syntax

Cnv_QueUserData (*conveyorNumber* [, *index*])

Parameter

conveyorNumber	Integer-Ausdruck, welcher die Conveyornummer repräsentiert.
index	Optional. Integer-Ausdruck, welcher den Index der Objektnummer im Queue definiert.

Rückgabewerte

Real-Wert

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueUserData Anweisung

Beispiel einer Cnv_QueUser Data Funktion

```
Real userData

userData = Cnv_QueUserData(1, 1)
```

Cnv_RobotConveyor Funktion

F

Gibt den Conveyor aus, der von einem Roboter verfolgt wird.

Syntax

Cnv_RobotConveyor [(*robotNumber*)]

Parameter

robotNumber Integer-Ausdruck, welcher die Roboternummer repräsentiert.

Rückgabewerte

Integer Conveyornummer. 0 = es wird kein Conveyor verfolgt.

Beschreibung

Werden mehrere Roboter benutzt, so kann Cnv_RobotConveyor verwendet werden, um zu sehen, welcher Conveyor aktuell von einem Roboter verfolgt wird.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_MakePoint Anweisung

Beispiel einer Cnv_RobotConveyor Funktion

```
Integer cnvNum

cnvNum = Cnv_RobotConveyor(1)
```

Cnv_Speed Funktion

F

Gibt die aktuelle Geschwindigkeit eines Conveyors aus.

Syntax

Cnv_Speed (*conveyorNumber*)

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

Rückgabewerte

Real-Wert in Millimetern pro Sekunde.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_Pulse

Beispiel einer Cnv_Speed Anweisung

```
Print "Conveyor speed: ", Cnv_Speed(1)
```

Cnv_Trigger Anweisung

S

Speichert den aktuellen Zustand der Conveyor-Position für die nächste Cnv_QueAdd Anweisung.

Syntax

Cnv_Trigger *conveyorNumber*

Parameter

conveyorNumber Integer-Ausdruck, welcher die Conveyornummer repräsentiert.

Beschreibung

Cnv_Trigger ist ein Software-Triggerbefehl, der verwendet werden muss, wenn kein Hardware-Trigger existiert, der mit der PG-Platine für den Conveyor-Encoder verkabelt ist.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Conveyor Tracking” installiert ist.

Verwandte Befehle

Cnv_QueAdd

Beispiel einer Cnv_Trigger Anweisung

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
  VGet FindParts.Part.CameraXYU(i), found, x, y, u
  Cnv_QueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cont Anweisung

S

Der **Cont**-Befehl setzt einen Task nach Ausführung einer Pausen-Anweisung wieder fort, oder wenn eine Pause durch die Schutzabschrankung hervorgerufen wurde.

Syntax

Cont

Beschreibung

Verwenden Sie den **Cont**-Befehl, um alle Tasks wieder fortzusetzen, die durch die Pausen-Anweisung oder die Schutzabschrankung unterbrochen wurden.

Verwandte Befehle

Pause

Beispiel einer Cont Anweisung

```
Function main
    Xqt 2, monitor, NoPause
    Do
        Jump P1
        Jump P2
    Loop
Fend

Function monitor
    Do
        If Sw(pswitch) = On then
            Pause
            Wait Sw(pswitch) = Off
            Cont
        End If
    Loop
Fend
```

Copy Befehl

>

Kopiert eine Datei an einen anderen Ort.

Syntax

Copy *source*, *destination*

Parameter

source	Pfad- und Dateiname des Ursprungsortes der zu kopierenden Datei(en).
destination	Pfad- und Dateiname des Zielortes, an den die angegebenen Ursprungsdateien kopiert werden sollen.

Beschreibung

Kopiert den angegebenen *source*-Dateinamen zum angegebenen *destination*-Dateinamen.

Wenn der Zielort bereits existiert, tritt ein Fehler auf.

Platzhalter (*, ?) sind in den definierten Dateinamen nicht gestattet.

Derselbe Pfad- und Dateiname darf nicht für Ursprungs- und Zieldateien definiert werden.

Wenn sie im Online-Fenster verwendet werden, kann man Anführungsstriche und Kommata auslassen.

Verwandte Befehle

ChDir, Dir, MkDir

Beispiel eines Copy Befehls

```
Function BackupData
    Copy "c:\data\test.dat", "c:\databack\test.dat"
Fend
```

Das folgende Beispiel wird vom Online-Fenster aus ausgeführt.

```
> copy test.dat test2.dat
```

Cos Funktion

F

Gibt den Kosinus eines numerischen Ausdrucks aus.

Syntax
Cos(*number***)**
Parameter
number Numerischer Ausdruck in Radianten.
Rückgabewerte
Numerischer Wert in Radianten, die den Kosinus des numerischen Ausdrucks *number* repräsentiert.
Beschreibung
Cos gibt den Kosinus des numerischen Ausdrucks aus. Der numerische Ausdruck (*number)* muss in Radianteneinheiten angegeben werden. Der von der **Cos** Funktion ausgegebenen Wert bewegt sich zwischen -1 und 1.

Verwenden Sie die DegToRad -Funktion, um Gradzahlen in Radiantenwerte umzuwandeln.

Verwandte Befehle
Abs, Atan, Atan2, Int, Mod, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Beispiel einer Cos Funktion

Das folgende Beispiel zeigt ein einfaches Programm, das den **Cos**-Befehl nutzt.

```
Function costest
   Real x
   Print "Please enter a value in radians"
   Input x
   Print "COS of ", x, " is ", Cos(x)
Fend
```

Die folgenden Beispiele verwenden die **Cos**-Funktion vom Online-Fenster aus.

Anzeige des Kosinus von 0.55:

```
>print cos(0.55)
 0.852524522059506
>

Display cosine of 30 degrees:

>print cos(DegToRad(30))
 0.866025403784439
>
```

Cp Anweisung

S

Stellt den CP- (Continuous Path) Bewegungsmodus ein.

Syntax

CP { **On | Off** }

Parameter

On | Off Das Schlüsselwort On (Ein) wird verwendet, um den CP-Modus aktivieren. Das Schlüsselwort Off (Aus) wird verwendet, um den CP-Modus zu deaktivieren.

Beschreibung

Der CP- (Continuous Path) Bewegungsmodus kann für die Roboter-Bewegungsbefehle Arc, Go, und Move verwendet werden. Wenn CP aktiviert ist, gestatten Arc, Go, und Move die Ausführung der nächsten Anweisung bei Beginn der Verzögerung.

Verwandte Befehle

CP Funktion, Arc, Move, Go

Beispiel einer CP-Anweisung

```
CP On
Move P1
Move P2
CP Off
```

Cp Funktion

F

Gibt den Status einer CP-Bewegung aus.

Syntax

CP

Rückgabewerte

0 = CP-Bewegung aus, 1 = CP-Bewegung ein.

Verwandte Befehle

CP Anweisung

Beispiel einer CP Funktion

```
If CP = Off Then
    Print "CP is off"
EndIf
```

Ctr Funktion

F

Gibt den Zählerwert des angegebenen Zählereingangs aus.

Syntax

Ctr(*bitNumber*)

Parameter

bitNumber Nummer des Hardwareeingangs, als Zähler eingestellt. Es können gleichzeitig nur 16 Zähler aktiv sein.

Rückgabewerte

Der aktuelle Zählstand des angegebenen Zählereingangs.

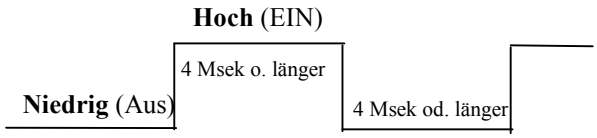
Beschreibung

Ctr arbeitet mit der CTReset-Anweisung, um es Hardwareeingängen zu ermöglichen, als Zähler zu funktionieren.

Jedes Mal, wenn ein Hardwareeingang, der als Zähler definiert ist, vom Off- (Aus) zum On- (Ein) Status geschaltet wird, erhöht dieser Eingang den Zählerwert um 1.

Die **Ctr**-Funktion kann jederzeit verwendet werden, um den aktuellen Zählerwert für jegliche Zählereingänge zu erhalten. Jeglicher Hardwareeingang kann als Zähler verwendet werden. Es können jedoch nur 16 Zähler gleichzeitig aktiv sein.

Zähler-Pulse Eingabe-Zeitkurve



Verwandte Befehle

CTReset

Beispiel einer Ctr Funktion

Das folgende Beispiel zeigt ein Code-Muster, das verwendet werden könnte, um den Wert eines Zählereingangs zu erhalten.

```
CTReset 3 'Setzt den Zähler für Eingang 3 auf 0 zurück
On 0      'Schaltet einen Ausgangsschalter ein

Do While Ctr(3) < 5
Loop
Off 0     'Wenn 5 Eingabezyklen für Eingang 3 gezählt wurden,
          'wird ausgeschaltet (Ausgang 0 aus)
```

CTReset Anweisung

> S

Setzt den Zählerwert des angegebenen Zählereingangs zurück. (Stellt auch den Hardwareeingang als Zählereingang ein.)

Syntax

CTReset(*bitNumber*)

Parameter

bitNumber	Nummer des Hardwareeingangs, als Zähler eingestellt. Die Nummer muss ein Integer-Ausdruck sein, der einen gültigen Hardwareeingang repräsentiert. Es können gleichzeitig nur 16 Zähler aktiv sein.

Beschreibung

CtrReset arbeitet mit der CTR-Funktion, um es Hardwareeingängen zu ermöglichen, als Zähler zu funktionieren. CTReset stellt den angegebenen Eingang als Zähler ein und startet den Zähler dann. Wird der angegebenen Eingang bereits als Zähler genutzt, so wird er zurückgesetzt und neu gestartet.

Hinweise

Ausschalten des Stroms und die Auswirkungen auf die Zähler:

Das Abschalten der Versorgungsspannung gibt alle Zähler frei.

Verwendung der Ctr Funktion:

Die Ctr Funktion wird für den Erhalt der aktuellen Werte des Zählereingangs verwendet.

Verwandte Befehle

Ctr

Beispiel einer CTReset Anweisung

Das folgende Beispiel zeigt ein Code-Muster, das verwendet werden könnte, um den Wert eines Zählereingangs zu erhalten.

```
CTReset 3 'Setzt den Zähler 3 auf 0 zurück
On 0      'Schaltet einen Ausgangsschalter ein
Do While Ctr(3) < 5
Loop
Off 0     'Wenn 5 Eingabezyklen für Eingang 3 gezählt wurden,
           'wird ausgeschaltet (Ausgang 0 aus)
```

CtrlDev Befehl

S

Spezifiziert das aktuelle Steuergerät.

Syntax

CtrlDev [*device*]

Parameter

device **PC** - Der PC kann Programme starten. (Vorgabeeinstellung.)

Remote – Programme können mit der Remote-E/A-Steuerung ausgeführt werden.

Beschreibung

CtrlDev spezifiziert das Steuergerät.

PC	Ermöglicht dem PC, Programme zu starten, zu unterbrechen, fortzusetzen and anzuhalten.
Remote	Ermöglicht es externem Equipment unter Verwendung diskreter Ein- und Ausgänge Programme zu starten, zu unterbrechen und fortzusetzen.

Wenn kein Parameter spezifiziert ist, zeigt **CtrlDev** die aktuelle Einstellung an.

Hinweise

Der Vorgabewert von CtrlDev (und die Verwendung von Verinit):

Wird der Verinit-Befehl ausgeführt, so wird der **CtrlDev**-Wert auf seinen Vorgabewert initialisiert: PC.

Verwandte Befehle

CtrlDev Funktion

Beispiel einer CtrlDev Anweisung

Das folgende Beispiel zeigt die zur Einstellung des aktuellen Steuergerätes erforderliche Syntax. Gezeigt werden die vom Online-Fenster ausgeführten Befehle.

```
> ctrldev remote   ' Externe Geräte können Programme starten

>ctrldev
Remote
>
```

CtrlDev Funktion

F

Gibt die Nummer des aktuellen Steuergerätes aus.

Syntax
CtrlDev

Rückgabewerte

1	**PC**
2	**Remote**

Verwandte Befehle
CtrlDev Befehl

Beispiel einer CtrlDev Funktion

```
Print "The current control device is: ", CtrlDev
```

CU Funktion

F

Extrahiert den U-Achsen-(Ausrichtungsachsen)-Koordinatenwert eines Punktes.

Syntax

CU(*point*)

Parameter

point Punktausdruck.

Rückgabewerte

Gibt einen numerischen Koordinatenwert aus, der die U-Achsen-Position repräsentiert. Der ausgegebene Wert ist eine Real-Zahl.

Beschreibung

Wird verwendet, um den individuellen Koordinatenwert für die U-Achse eines spezifischen Punktes zu erhalten.

Um die aktuelle U-Achsen-Position des Roboters zu erhalten verwenden Sie bitte **P*** oder **Here** für das Punktargument.

Verwandte Befehle

CX, CY, CZ

CU Beispiel

Das folgende Beispiel extrahiert den U-Achsen-Koordinatenwert von Punkt P100 und übergibt den Koordinatenwert an die Variable *u*.

```
Function test
   Real u
   u = CU(P100)
   Print "The U Axis Coordinate of P100 is ", u
Fend
```

CU Anweisung

S

Stellt die U-Koordinate eines Punktes ein.

Syntax
CU(*point***) =** *value*

Parameter

point	**P***number* oder **P**(*expr*) oder Punktlabel.
value	Real-Ausdruck, der den neuen Koordinatenwert repräsentiert.

Verwandte Befehle
CU Funktion

Beispiel einer CU Anweisung

```
CU(pick) = 45.0
```

CurDir$ Funktion

F

Gibt eine Zeichenkette aus, die den aktuellen Pfad repräsentiert.

Syntax

CurDir$

Rückgabewerte

Eine Zeichenkette, die das aktuelle Laufwerk und den aktuellen Pfad beinhaltet.

Verwandte Befehle

ChDir, CurDrive$

Beispiel einer CurDir$ Funktion

```
Print "The current directory is: ", CurDir$
```

CurDrive$ Funktion

F

Gibt eine Zeichenkette aus, die das aktuelle Laufwerk repräsentiert.

Syntax

CurDrive$

Rückgabewerte

Eine Zeichenkette, die den aktuellen Laufwerksbuchstaben enthält.

Verwandte Befehle

ChDrive, CurDir$

Beispiel einer CurDrive$ Function

```
Print "The current drive is: ", CurDrive$
```

Curve Anweisung

S

Definiert die Daten und Punkte, die notwendig sind, um einen Arm an einem geschwungenen Pfad entlang zu bewegen. Viele Datenpunkte können im Pfad definiert werden, um die Präzision des Pfades zu steigern.

Syntax

Curve *fileName, closure, mode, numAxes, pointList*

Parameter

Hinweise zum Dateinamen Ein Zeichenkettenausdruck für Pfad und Namen der Datei, in welcher die Punktdaten gespeichert sind. Dem angegebenen *fileName* wird die Dateinamenerweiterung CRV angehängt, so dass keine Dateinamenerweiterung vom Anwender spezifiziert werden muss. Wenn der **Curve**-Befehl ausgeführt wird, wird *fileName* erzeugt.

closure Spezifiziert, ob die definierte **Curve** am Ende der Kurvenbewegung geschlossen wird, oder offen bleibt. Dieser Parameter muss auf einen von zwei möglichen Werten eingestellt werden, wie im Folgenden angezeigt.

C – Geschlossenen Kurve (Closed Curve)

O – Offenen Kurve (Open Curve)

Wenn die offene Kurve spezifiziert wird, erzeugt der **Curve**-Befehl die notwendigen Daten, um den Arm am letzten Punkt der spezifizierten Punktserie zu stoppen. Wird die geschlossene Kurve spezifiziert, erzeugt der **Curve**-Befehl die notwendigen Daten, um die Bewegung durch den spezifizierten Endpunkt hindurch fortzusetzen und die Bewegung nach Rückkehr des Armes zur Startposition der angegebenen Punktserie für den **Curve**-Befehl zu stoppen.

mode Spezifiziert, ob der Arm automatisch in Tangentenrichtung der U-Achse interpoliert wird, oder nicht.

Modus-Einstellung	Tangential-korrektur
0	Nein
2	Ja

Wenn eine Tangentialkorrektur spezifiziert wird, verwendet der **Curve**-Befehl nur die U-Achsen-Koordinate des Startpunktes in der Punktserie. Die Tangentialkorrektur behält fortlaufend die Ausrichtung der Werkzeugtangente der Kurve in der XY-Ebene bei. Sie wird spezifiziert, wenn Werkzeuge wie z.B. Abschneidevorrichtungen installiert werden, die fortlaufender Tangentenausrichtung bedürfen. Wird (mit dem Parameter *closure*) eine geschlossenen Kurve spezifiziert, mit automatischer Interpolation in der Tangentenrichtung der U-Achse, so rotiert die U-Achse von ihrem Startpunkt aus um 360 °. Es ist daher notwendig, vor Ausführung des CVMove-Befehls mit dem Range-Befehl den Bewegungsbereich der U-Achse einzustellen, damit die 360°-Rotation der U-Achse keinen Fehler hervorruft.

numAxes Integer Nummer zwischen 2 und 4, welche die Anzahl der in der Kurvenbewegung gesteuerten Achsen wie folgt spezifiziert:

2 – Generiert eine Kurve in der XY-Ebene ohne Bewegung der Z-Achse oder Rotation der U-Achse.

3 - Generiert eine Kurve in der XYZ-Ebene ohne Rotation der U-Achse. (Theta 1, Theta2, und Z)

4 - Generiert eine Kurve in der XYZ-Ebene mit Rotation der U-Achse. (Steuert alle 4 Achsen)

Die Achsen, die nicht zur Steuerung während des **Curve**-Befehls angewählt wurden, halten ihre vorherigen Codiererpulslagen bei und bewegen sich während der **Curve** Bewegung nicht.

pointList { Punktausdruck | P(*start*:*finish*) } [, *output command*] ...
Dieser Parameter besteht in der Realität aus einer Serie von Punktnummern und optionalen Ausgangsanweisungen, die entweder durch Kommata voneinander getrennt sind oder aus einem aufsteigenden Bereich von Punkten, getrennt durch Doppelpunkt. Normalerweise sind die Punktserien durch Kommata voneinander getrennt, wie im folgenden gezeigt:

```
Curve "MyFile", O, 0, 4, P1, P2, P3, P4
```

Manchmal definiert der Anwender eine Punktserie unter Verwendung eines aufsteigenden Bereichs von Punkten, wie unten dargestellt.

```
Curve "MyFile", O, 0, 4, P(1:4)
```

Im oben dargestellten Beispiel hat der Anwender eine Kurve mit den Punkten P1, P2, P3, und P4 definiert. Der *output command* ist optional und wird verwendet, um den Ausgabebetrieb während der Kurvenbewegung zu steuern. Für digitale Ausgänge und Merker können die Befehle On oder Off verwendet werden. Die Eingabe eines Ausgabebefehls nach einer Punktnummer in der Punktserie veranlasst die Ausführung des *output*-Befehls, wenn der Arm den Punkt unmittelbar vor dem *output*-Befehl erreicht. Eine **Curve**-Anweisung darf maximal 16 Ausgangsbefehle enthalten. Im folgenden Beispiel wird der "On 2"–Befehl ausgeführt, sobald der Arm den Punkt P2 erreicht, dann fährt der Arm zu allen Punkten zwischen und inklusive P3 und P10.

```
Curve "MyFile", C, 0, 4, P1, P2, ON 2, P(3:10)
```

Beschreibung

Der **Curve**-Befehl erzeugt Daten, die den Manipulatorarm die durch die Punkstserie *pointList* definierte Kurve entlang bewegen und speichert die Daten im Dateispeicher der Robotersteuerung. Der CVMove-Befehl verwendet die von **Curve** erzeugten Daten, um den Manipulator in einer Art CP-Bewegung zu bewegen.

Die erzeugten Daten sind **Curve-**Informationen, definiert im orthogonalen Koordinatensystem. Es ist nicht notwendig, vor der Ausführung des **Curve**-Befehls Geschwindigkeiten oder Beschleunigungen festzulegen. Die Parameter für Armgeschwindigkeit und –Beschleunigung können vor Ausführen von CVMove jederzeit mittels der Befehle Speeds oder Accels geändert werden.

Durch die Befehle Base oder Local definierte Punkte können in Serie verwendet werden, um die Kurve an der gewünschten Position zu lokalisieren. Die Punkte können, wenn sie zum lokalen Koordinatensystem gehören, durch einen dem **Curve**-Befehl folgenden Local-Befehl geändert werden. Definieren Sie dazu alle spezifizierten Punkte in der Punktserie für den **Curve**-Befehl als Punkte mit Base- oder Local- Attributen.

Hinweis

Verwenden Sie nach Möglichkeit die Tangentialkorrektur

Es wird empfohlen, immer wenn möglich, die Tangentialkorrektur zu verwenden, insbesondere, wenn CVMove in einer kontinuierlichen Schleife durch dieselben Punkte verwendet wird. Wenn Sie keine Tangentialkorrektur verwenden, ist es möglich, dass der Roboter bei höheren Geschwindigkeiten nicht dem korrekten Pfad folgt.

Minimale und maximale Anzahl erlaubter Punkte in der offenen Kurve (Open Curve)

Offene Kurven können unter Verwendung von 3 bis 200 Punkten definiert werden.

Minimale und maximale Anzahl erlaubter Punkte in der geschlossenen Kurve (Closed Curve)

Geschlossene Kurven können unter Verwendung von 3 bis 50 Punkten definiert werden.

Die Modi 1 und 3 werden in SPEL$^+$ nicht mehr benötigt

Die Modi 1 und 3 wurden in der SPEL Sprache in SRC-3xx Steuerungen verwendet, um festzulegen, ob nach Erreichen des letzten Punktes eine Verzögerung einsetzen sollte. In SPEL$^+$ wurde dieses Feature durch den CP-Parameter für die CVMove-Anweisung ersetzt.

Potentielle Fehler

Versuch, den Arm außerhalb des Arbeitsbereiches zu bewegen

Der **Curve-**Befehl kann den Bewegungsbereich für den definierten Kurvenpfad nicht prüfen. Dies bedeutet, dass ein vom Anwender definierter Pfad den Roboterarm dazu veranlassen dürfte, sich außerhalb seines normalen Arbeitsbereiches zu bewegen. In diesem Fall tritt ein “out of range”-(„außerhalb des Arbeitsbereiches“-) Fehler auf.

Verwandte Befehle

Accels Funktion, Arc, CVMove, Move, Speeds

Beispiel einer Curve Anweisung

Das folgende Beispiel bestimmt den Dateinamen der Daten der freien Kurve als MYCURVE.CRV, erzeugt eine Kurve, die die Punkte P1 bis P7 entlang fährt, schaltet bei P2 den Ausgangsport 2 EIN, und verzögert den Arm bei P7.

Erstellen der Kurve

```
> curve "mycurve", 0, 0, 4, P1, P2, On 2, P(3:7)
```

Bewegen des Armes zu P1 in einer geraden Linie

```
> jump P1
```

Bewegen des Armes nach der “Mycurve” genannten Kurvendefinition.

```
> cvmove "mycurve"
```

CVMove Anweisung

S

Führt die kontinuierliche Spline-Pfad-Bewegung aus, die durch den **Curve**-Befehl definiert ist.

Syntax

CVMove *fileName* [**CP**] [*searchExpr*]

Parameter

fileName	Zeichenkettenausdruck für Pfad und Dateinamen, die für die CP-Bewegungsdaten zu verwenden sind. Diese Datei muss im Vorangegangenen bereits durch den Curve-Befehl erzeugt worden und auf einer PC-Festplatte gespeichert sein.
CP	Optional. Spezifiziert die CP-Bewegung nach dem letzten Punkt.
searchExpr	Optional. Ein Till oder Find -Ausdruck. **Till** | **Find** **Till Sw**(*expr*) = {**Ein** | **Aus**} **Find Sw**(*expr*) = {**Ein** | **Aus**}

Beschreibung

CVMove führt die kontinuierliche Spline-Pfad-Bewegung aus, definiert durch die Daten in der Datei *fileName*, die sich auf der Festplatte des PCs befinden. Diese Datei muss im Vorangegangenen bereits durch den Curve-Befehl erzeugt worden sein. Es sollten keine Disketten oder andere langsame Medien verwendet werden, da sonst eine Verzögerung beim Laden der Daten aus der Datei entsteht.

Im System können gleichzeitig multiple Dateien existieren. Ist keine Dateinamenerweiterung angegeben, wird von der Endung CRV ausgegangen.

Der Anwender kann mit den Befehlen Speeds und Accels Geschwindigkeit und Beschleunigung für die CP-Bewegung für **CVMove** ändern.

Wenn der Curve-Befehl im Vorangegangenen bereits unter Verwendung von Punkten mit Local-Definitionen ausgeführt wurde, können Sie die Arbeitsposition mithilfe des Local-Befehls ändern.

Bevor Sie den **CVMove**-Befehl ausführen, stellen Sie sicher, dass sich der gesamte Bewegungspfad innerhalb des Bewegungsbereichs des Manipulatorarms befindet. Wenn der Pfad bestimmt wird, prüft das System nicht, ob sich der definierte Pfad innerhalb des Bewegungsbereiches befindet, oder nicht.

Hinweise

Der neue Befehlsname lautet CVMove:

In älteren SPEL Versionen hieß dieser Befehl CVMOV, wurde aber später zu **CVMove** modifiziert, da die meisten Anwender den **CVMove** Befehlsnamen besser identifizieren konnten.

Verwandte Befehle

Accels Funktion, Arc, Curve, Move, Speeds, Till, TillOn

Beispiel einer CVMove Anweisung

Das folgende Beispiel bestimmt den Dateinamen der Daten der freien Kurve als MYCURVE.CRV, erzeugt eine Kurve, die die Punkte P1 bis P7 entlang fährt, schaltet bei P2 den Ausgangsport 2 EIN, und verzögert den Arm bei P7.

Erstellen der Kurve

```
> curve "mycurve", 0, 0, 4, P1, P2, On 2, P(3:7)
```

Bewegen des Armes zu P1 in einer geraden Linie

```
> jump P1
```

Bewegen des Armes nach der “Mycurve” genannten Kurvendefinition.

```
> cvmove "mycurve"
```

CurPos Funktion

F

Gibt die aktuelle Position des spezifizierten Roboters aus.

Syntax

CurPos [(*robotNumber*)]

Parameter

robotNumber	Optional. Definiert, für welchen Roboter Positionsdaten ausgegeben werden sollen. Wird sie weggelassen, wird die aktuelle Position des aktuellen Roboters ausgegeben.

Rückgabewerte

Ein Roboterpunkt, der die aktuelle Position des angegebenen Roboters spezifiziert.

Verwandte Befehle

InPos, FindPos

Beispiel einer CurPos Funktion

```
Function main

   Xqt showPosition
   Do
      Jump P0
      Jump P1
   Loop
Fend

Function showPosition

   Do
      P99 = CurPos
      Print CX(P99), CY(P99)
   Loop
Fend
```

Cx FunktionXXX

F

Extrahiert den X-Achsen-Koordinatenwert eines Punktes.

Syntax

CX(*point***)**

Parameter

point Punktausdruck.

Rückgabewerte

Gibt einen numerischen Koordinatenwert aus, der die X-Achsen-Position repräsentiert. Der ausgegebene Wert ist eine Real-Zahl.

Beschreibung

Wird verwendet, um den individuellen Koordinatenwert für die X-Achse eines spezifischen Punktes zu erhalten.

Um die aktuelle X-Achsen-Position des Roboters zu erhalten, verwenden Sie bitte **P*** oder **Here** für den Punktparameter.

Verwandte Befehle

CU, CY, CZ

Beispiel einer CX Funktion

Das folgende Beispiel extrahiert den X-Achsen-Koordinatenwert von Punkt „pick“ und übergibt den Koordinatenwert an die Variable *x*.

```
Function cxtest
   Real x
   x = CX(pick)
   Print "The X Axis Coordinate of point 'pick' is", x
Fend
```

CX Anweisung

S

Stellt die X-Koordinate eines Punktes ein.

Syntax

CX(*point***)** = *value*

Parameter

point	**P***number* oder **P**(*expr*) oder Punktlabel.
value	Real-Ausdruck, der den neuen Koordinatenwert repräsentiert.

Verwandte Befehle

CX Funktion

Beispiel einer CX Anweisung

```
CX(pick) = 25.34
```

CY Funktion

F

Extrahiert den Y-Achsen-Koordinatenwert eines Punktes.

Syntax
CY(*point***)**
Parameter
point Punktausdruck.

Rückgabewerte
Gibt einen numerischen Koordinatenwert aus, der die Y-Achsen-Position repräsentiert. Der ausgegebene Wert ist eine Real-Zahl.

Beschreibung
Wird verwendet, um den individuellen Koordinatenwert für die Y-Achse eines spezifischen Punktes zu erhalten.

Um die aktuelle Y-Achsen-Position des Roboters zu erhalten, verwenden Sie bitte **P*** oder **Here** für den Punktparameter.

Verwandte Befehle
CU, CX, CZ

Beispiel einer CY Funktion

Das folgende Beispiel extrahiert den Y-Achsen-Koordinatenwert von Punkt „pick“ und übergibt den Koordinatenwert an die Variable *y*.

```
Function cytest
   Real y
   y = CY(pick)
   Print "The Y Axis Coordinate of point 'pick' is", y
Fend
```

CY Anweisung

S

Stellt die Y-Koordinate eines Punktes ein.

Syntax

CY(*point***)** = *value*

Parameter

point	**P***number* oder **P**(*expr*) oder Punktlabel.
value	Real-Ausdruck, der den neuen Koordinatenwert repräsentiert.

Verwandte Befehle

CY Funktion

Beispiel einer CY Anweisung

```
CY(pick) = 33.3
```

CZ Funktion

F

Extrahiert den Z-Achsen-Koordinatenwert eines Punktes.

Syntax

CZ(*point***)**

Parameter

point Punktausdruck.

Rückgabewerte

Gibt einen numerischen Koordinatenwert aus, der die Z-Achsen-Position repräsentiert.

Beschreibung

Wird verwendet, um den individuellen Koordinatenwert für die Z-Achse eines spezifischen Punktes zu erhalten.

Um die aktuelle Z-Achsen-Position des Roboters zu erhalten, verwenden Sie bitte **P*** oder **Here** für den Punktparameter.

Verwandte Befehle

CU, CX, CY

Beispiel einer CZ Funktion

Das folgende Beispiel extrahiert den Z-Achsen-Koordinatenwert von Punkt „pick“ und übergibt den Koordinatenwert an die Variable *z*.

```
Function cztest
   Real z
   z = CZ(pick)
   Print "The Z Axis Coordinate of point 'pick' is", z
Fend
```

CZ Anweisung

S

Stellt die Z-Koordinate eines Punktes ein.

Syntax

CZ(*point***)** = *value*

Parameter

point	**P***number* oder **P**(*expr*) oder Punktlabel.
value	Real-Ausdruck, der den neuen Koordinatenwert repräsentiert.

Verwandte Befehle

CZ Funktion

Beispiel einer CZ Anweisung

```
CZ(pick) = -30.0
```

Date Anweisung

>

Spezifiziert das aktuelle Datum in der Steuerung und zeigt es an.

Syntax

Date [*yyyy, mm, dd*]

Parameter

yyyy	Integer-Ausdruck für eine Jahreszahl.
mm	Integer-Ausdruck für einen Monat.
dd	Integer-Ausdruck für einen Tag.

Rückgabewerte

Wenn der **Date**-Befehl ohne jegliche Parameter eingegeben wird, wird das aktuelle Datum angezeigt .

Beschreibung

Spezifiziert das aktuelle **Date** (Datum) für die Steuerung. Dieses Datum wird für die Dateien in der Steuerung verwendet. Alle in der Steuerung befindlichen Daten sind mit einem Datum versehen. **Date** berechnet automatisch den Tag der Woche für die **Date-**Anzeige.

Verwandte Befehle

Time, Date$

Beispiel der Date-Anweisung

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt.

```
> Date
1999/09/27

> Date 1999,10,1

> Date
1999/10/01
```

Date$ Funktion

F

Gibt das Systemdatum aus.

Syntax

Date$

Rückgabewerte

Eine Zeichenkette, die das Datum im folgenden Format beihaltet: *yyyy/mm/dd*.

Beschreibung

Date$ wird verwendet, um das Datum des Steuerungssystems in einer Programmanweisung zu erhalten. Sie müssen die Date-Anweisung verwenden, um das Systemdatum einzustellen.

Verwandte Befehle

Date, GetDate, Time, Time$

Beispiel einer Date$ Funktion

```
Function LogErr(errNum As Integer)

    AOpen "errlog.dat" As #30
    Print #30, "Error ", errNum, " ", ErrMsg$(errNum), " ", Date$, " ",
Time$
    Close #30
Fend
```

Declare Anweisung

S

Definiert eine externe Funktion in einer Dynamic Link Library (DLL).

Syntax

Declare *funcName, "dllFile", "alias"* [, (*argList*)] **As** *type*

Parameter

funcName	Der Name der Funktion, so wie sie von Ihrem Programm aufgerufen wird.
dllFile	Pfad und Name der Library-Datei. Dabei muss es sich um eine Buchstabensymbol-Zeichenkette handeln (Zeichen durch Anführungsstriche begrenzt). Sie können ebenfalls ein durch #define definiertes Makro verwenden.
alias	Der tatsächliche Name der Funktion in der DLL. Er muss aus einer Buchstabensymbol-Zeichenkette bestehen und es wird Groß-und Kleinschreibung unterschieden (Zeichen durch Anführungsstriche begrenzt). Sie können ebenfalls ein durch #define definiertes Makro verwenden.
arglist	Optional. Liste der DLL Argumente. Siehe Syntax weiter unten.
type	Erforderlich. Sie müssen den Funktionstyp definieren.

Das arglist-Argument hat die folgende Syntax:

[{**ByRef | ByVal**}] *varName* [( )] **As** *type*

ByRef	Optional. Spezifizieren Sie ByRef, wenn Sie möchten, dass Veränderungen im Variablenwert von der Calling-Funktion gesehen werden.
ByVal	Optional. Spezifizieren Sie ByVal, wenn Sie *nicht* möchten, dass Veränderungen im Variablenwert von der Calling-Funktion gesehen werden. Dies ist die Vorgabeeinstellung.
varName	Erforderlich. Name der Variable, welche das Argument repräsentiert; folgt den Standardkonventionen für Variablenbenennung. Matrizen können für alle Datentypen außer String weitergereicht werden und müssen ByRef sein.
type	Erforderlich. Der Argumenttyp muss definiert werden.

Beschreibung

Verwenden Sie die Declare-Anweisung, um DLL-Funktionen aus einem beliebigen Programm in der aktuellen Programmgruppe aufzurufen. Die Declare-Anweisung muss außerhalb von Funktionen verwendet werden.

Numerische Variablen durch ByVal weitergeben

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a As Long) As Long
VC++  long _stdcall MyDllFunc(long a);
```

Zeichenkettenvariablen durch ByVal weitergeben

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a$ As String) As Long
VC++  long _stdcall MyDllFunc(char *a);
```

Numerische Variablen durch ByRef weitergeben

```
SPEL:  Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a As Long) As
Long
VC++   long _stdcall MyDllFunc(long *a);
```

Zeichenkettenvariablen durch ByRef weitergeben

```
SPEL:  Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a$ As String) As
Long
VC++   long _stdcall MyDllFunc(char *a);
```

Wenn Sie eine Zeichenkette mithilfe von ByRef weitergeben, können Sie die Zeichenkette in der DLL ändern. Die maximale Länge der Zeichenkette beträgt 256 Zeichen. Sie müssen sicherstellen, dass die maximale Länge nicht überschritten wird.

Sie müssen ebenfalls sicherstellen, dass der Zeichenkette Platz zugeordnet wird, bevor die DLL aufgerufen wird. Es ist das Beste, über die Space$ Funktion 256 Bytes zuzuordnen, wie im folgenden Beispiel gezeigt.

```
Declare ChangeString, "mystuff.dll", "ChangeString", (ByRef a$ As String) As
Long

    Function main
        String a$

        'Weist der Zeichenkette ein Leerzeichen zu, bevor die DLL aufgerufen
    wird.
        a$ = Space$(256)
        Call ChangeString(a$)
        Print "a$ after DLL call is: ",  a$
    Fend
```

Numerische Matrizen durch ByRef weitergeben

```
SPEL:  Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a() As Long) As
Long
VC++   long _stdcall MyDllFunc(long *a);
```

Ausgabe von Werten von der DLL Funktion

Die DLL-Funktion kann einen Wert für jeglichen Datentyp ausgeben, inklusive String (Zeichenketten). Für eine Zeichenkette muss jedoch ein Datenzeiger an eine der DLL zugeordnete Zeichenkette ausgegeben werden. Und der Funktionsname muss in einem Dollarzeichen enden, so wie bei allen SPEL Zeichenkettenvariablen und Funktionen. Bitte beachten Sie, dass ‘alias’ kein Dollarzeichen-Suffix hat.

Zum Beispiel:

```
    Declare ReturnLong, "mystuff.dll", "ReturnLong", As Long
    Declare ReturnString$, "mystuff.dll", "ReturnString", As String

    Function main

        Print "ReturnLong = ", ReturnLong()
        Print "ReturnString$ = ", ReturnString$()
    Fend
```

Verwandte Befehle

Function...Fend

Beispiel einer Declare Anweisung

```
Declare MyDLLTest, "mystuff.dll", "MyDLLTest" As Long

Function main
    Print MyDLLTest
Fend

' Deklariert eine DLL-Funktion mit zwei Integer Argumenten
' und verwendet ein #define, um den DLL-Dateinamen festzulegen

#define MYSTUFF "mystuff.dll"

Declare MyDLLCall, MYSTUFF, "MyTestFunc", (var1 As Integer, var2 As
Integer) As Integer
```

DegToRad Function

> F

Konvertiert Gradmaße zu Radianten.

Syntax

DegToRad(*degrees***)**

Parameter

degrees Real-Ausdruck, welcher die Gradzahlen repräsentiert, die in Radianten konvertiert werden sollen.

Rückgabewerte

Ein Double-Wert, der die Zahl des Radianten enthält.

Verwandte Befehle

ATan, ATan2, RadToDeg Function

Beispiel einer DegToRad Funktion

```
s = Cos(DegToRad(x))
```

Del Befehl

>

Löscht eine oder mehrere Dateien.

Syntax

Del *fileName*

Parameter

fileName	Pfad und Name der zu löschenden Datei(en). Der Dateiname sollte mit einer Dateinamenerweiterung spezifiziert werden.

Beschreibung

Löscht die angegebene(n) Datei(en) von der Festplatte des PCs.

Dateinamen und Dateinamenerweiterungen dürfen Platzhalter wie (* oder ?) enthalten. Wird *.* an Stelle eines Dateinamens eingegeben, erscheint die folgende Nachricht:

Are You Sure(Y/N) ?

Um alle Dateien im aktuellen Verzeichnis zu löschen, treffen Sie folgende Wahl: Y
Um den **Del**-Befehl abzubrechen, treffen Sie folgende Wahl: N

Verwandte Befehle

Kill

Del Beispiel

Die folgenden Beispiele werden mittels des **Del**-Befehls vom Online-Fenster aus getätigt:

```
> Del *.pnt  'Löscht alle Punktdatein aus dem aktuellen Verzeichnis
> Del \BAK\*.*  'Löscht alle Dateien aus dem BAK-Unterverzeichnis
```

Dir Befehl

>

Zeigt den Inhalt des angegebenen Verzeichnisses an.

Syntax

Dir [*filename*]

Parameter

fileName	Pfad und Name der im angegebenen Pfad zu suchenden Datei. Der Dateiname wird nur verwendet, um spezifische Dateien im angegebenen Verzeichnis anzuzeigen, anstatt alle Dateien anzuzeigen. Dateiname und Dateinamenerweiterung dürfen Platzhalter (*, ?) enthalten.

Rückgabewerte

Zeigt den Inhalt des angegebenen Verzeichnisses an.

Beschreibung

Zeigt Dateinamen, Verzeichnisnamen, Dateigröße und-Datum und die Zuschreibung von Datum und Uhrzeit für bestimmte Verzeichnisse und Dateien an. Für diejenigen Anwender, die bereits Erfahrungen mit DOS haben: der **Dir**-Befehl in SPEL funktioniert ähnlich dem dir-Befehl unter DOS.

Hinweise

Wenn Pfad oder Dateiname weggelassen werden:

- Wenn der Pfad weggelassen wird, zeigt **Dir** die Datei(en) im aktuellen Verzeichnis an.
- Wenn der Dateiname weggelassen wird, zeigt **Dir** alle Dateien im aktuellen Verzeichnis an. Dies entspricht der Eingabe von *.* an Stelle eines Dateinamens.

Verwandte Befehle

ChDir, ChDrive

Beispiel eines Dir Befehls

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt.

```
> Dir

> Dir TEST.*

> Dir *.DAT
```

Dist Funktion

F

Gibt die Entfernung zwischen zwei Roboterpunkten aus.

Syntax

Dist (*point1*, *point2*)

Parameter

point1, point2 Spezifiziert zwei Roboter-Punktausdrücke (**P***, **P***number*, **P**(*expr*), Punktlabel).

Rückgabewerte

Gibt die Entfernung zwischen beiden Punkten aus (Real-Wert in Millimetern).

Verwandte Befehle

CU, CX, CY, CZ

Beispiel einer Dist Funktion

```
Real distance

distance = Dist(P1, P2)
```

Do...Loop Anweisung

S

Wiederholt einen Block von Anweisungen während eine Bedingung WAHR ist, oder bis eine Bedingung WAHR wird.

Syntax

Do [{ **While** | **Until** } *condition*]
    [*statements*]
[**Exit Do**]
    [*statements*]
Loop

Oder Sie können diese Syntax verwenden:

Do
    [*statements*]
[**Exit Do**]
    [*statements*]
Loop [{ **While** | **Until** } *condition*]

Die Do…Loop Anweisungssyntax besteht aus folgenden Teilen:

Teil	Beschreibung
condition (Bedingung)	Optional. Numerischer Ausdruck oder Zeichenkettenausdruck, der WAHR oder FALSCH ist. Wenn die *condition* (*Bedingung)* Null ist, wird sie als FALSCH behandelt.
statements	Eine oder mehrere Anweisungen werden wiederholt, während oder bis die *condition (Bedingung)* WAHR ist.

Beschreibung

Eine Anzahl von **Exit Do** Anweisungen kann an einem beliebigen Ort innerhalb von **Do...Loop** platziert werden, als alternative Art und Weise einen **Do...Loop** zu verlassen. **Exit Do** wird oft nach der Bewertung einer Bedingung verwendet, z.B., **If...Then**, in welchem Fall die **Exit Do** Anweisung die Steuerung an die direkt auf **Loop** folgende Anweisung überträgt.

Wenn **Exit Do** innerhalb verschachtelter **Do...Loop** Anweisungen verwendet wird, überträgt **Exit Do** die Steuerung an den Loop der sich eine Verschachtelungsebene über dem Loop befindet, in dem **Exit Do** auftritt.

Verwandte Befehle

For...Next, While...Wend

Beispiel einer Do-Anweisung

```
ROpen "test.dat" As #30
Do While Not Eof(30)
    Line Input #30, tLine$
    Print tLine$
Loop
Close #30
```

Double Anweisung

S

Deklariert Double-Variablen. (8 Byte Zahl).

Syntax

Double *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameter

varName	Variablenname, den der Anwender als **Double** deklarieren will.
subscripts	Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt **(***dim1* [, *dim2* [, *dim3*]]**)** *dim1*, *dim2*, *dim3* können eine ganze Zahl zwischen 0-32767 sein.

Beschreibung

Double wird verwendet, um Variablen als **Double** zu deklarieren. Alle Variablen sollten in einer Funktion ganz oben deklariert sein.

Verwandte Befehle

Boolean, Byte, Global, Integer, Long, Real, String

Beispiel einer Double-Anweisung

Das folgende Beispiel zeigt ein einfaches Programm, das unter Verwendung von **Double** einige Variablen deklariert.

```
Function doubletest
  Double var1
  Double A(10)         'Eindimensionale Feldvariable aus Doubles
  Double B(10, 10)     'Zweidimensionale Feldvariable aus Doubles
  Double C(10, 10, 10) 'Dreidimensionale Feldvariable aus Doubles
  Double arrayvar(10)
  Integer i
  Print "Please enter a Number:"
  Input var1
  Print "The variable var1 = ", var1
  For I = 1 To 5
    Print "Please enter a Number:"
    Input arrayvar(i)
    Print "Value Entered was ", arrayvar(i)
  Next I
Fend
```

EClr Anweisung

S

Löscht den Fehlerstatus nach Auftreten eines Fehlers.

Syntax

EClr

Beschreibung

EClr wird verwendet, um den Fehlerstatus nach Auftreten eines Fehlers zu löschen und wird normalerweise in einem Fehlerbehandlungs-Unterprogramm benutzt. Wenn im Normalfall ein Fehler auftritt und keine OnErr-Anweisung vorhanden ist (d.h. es wird keine Fehlerbehandlung benutzt), wird der Task abgebrochen und es wird der entsprechende Fehlercode angezeigt. Wenn jedoch eine Fehlerbehandlungsroutine verwendet wird, um den Fehler abzufangen (durch Verwendung des OnErr Befehls), muss der Fehler gelöscht werden, um eine erfolgreiche Rückkehr von der Fehlerbehandlungsroutine zu gestatten.

Einschränkungen

Denken Sie daran, dass alle Fehler mit der EClr-Anweisung gelöscht werden müssen:

Wenn der Anwender in der Fehlerbehandlungsunterroutine jedoch keine **EClr** Anweisung verwendet, um den Fehler zu löschen, kann das System in einer Endlosschleife stecken bleiben. Wenn der Fehler nicht mithilfe der Fehlerbehandlungsroutine zurückgesetzt wird, gibt das System auf den OnErr-Befehl hin die Steuerung sofort nach seiner Rückkehr zur Hauptroutine an das Fehlerbearbeitungsprogramm zurück. Der Grund dafür ist, dass das System davon ausgeht, es sei erneut ein Fehler aufgetreten, während es tatsächlich nur auf den vorangegangenen Fehler reagiert (der nicht gelöscht wurde). Da die Fehlerbehandlungsroutine den Fehler nicht zurücksetzt, tritt eine Endlosschleife auf. (Seien Sie daher vorsichtig!)

Verwandte Befehle

Err, OnErr, Return

Beispiel einer EClr Anweisung

Das folgende Beispiel zeigt ein einfaches Dienstprogramm, das prüft, ob die Punkte P0-P399 existieren. Wenn der Punkt nicht existiert, erscheint eine Nachricht auf dem Bildschirm, um den Anwender diesen Umstand wissen zu lassen. Das Programm verwendet den CX-Befehl, um jeden einzelnen Punkt dahin gehend zu testen, ob er definiert wurde, oder nicht. Wenn ein Punkt nicht definiert ist, wird die Steuerung an die Fehlerbehandlungsroutine übergeben und auf dem Bildschirm erscheint eine Nachricht, die dem Anwender mitteilt, welcher Punkt nicht definiert war.

```
Function eclrtest
   Integer i, errnum
   Real temp

   OnErr Goto eHandle
   For i = 0 To 399
     temp = CX(P(i))
   Next i
   End
'
'
'*****************************************
'* Error Handler                         *
'*****************************************
eHandle:
   errnum = Err(0); EClr
   'Überprüft, ob ein undefinierter Punkt verwendet wird
   If errnum = 78 Then
      Print "Point number P", i, " is undefined!"
   Else
      Print "ERROR: Error number ", errnum, " Occurred."
   End If
   EResume Next
Fend
```

ENetIO_AnaIn Funktion

Gibt den Status des angegebenen EtherNet I/O Analog-Ports aus.

Syntax

ENetIO_AnaIn(*channel*)

Parameter

channel Integer-Ausdruck oder I/O-Label, welches einen der analogen Eingangsports repräsentiert.

Rückgabewerte

Gibt einen Integer-Wert zwischen 0 und 255 aus. Der Rückgabewert beträgt 8 Bits, wobei jedes Bit einem analogen EtherNet I/O-Eingabekanal entspricht.

Beschreibung

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaOut, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_AnaIn Funktion

```
Print ENetIO_AnaIn(3)
```

ENetIO_AnaGetConfig Anweisung

Konfiguriert ein analoges Ethernet I/O Eingangs- oder Ausgangsmodul.

Syntax

ENetIO_AnaGetConfig (*channel, gain, offset, loScale, hiScale*)

Parameter

channel	Integer-Ausdruck oder I/O-Label, welches einen der analogen Eingangs- oder Ausgangsports repräsentiert.
gain	Realvariable zum Empfang der Eingangsverstärkung. 0 zeigt die Vorgabeeinstellung an.
offset	Realvariable zum Empfang des Eingangs-Offsets. 0 zeigt die Vorgabeeinstellung an.
loScale	Realvariable zum Empfang des niedrigsten Wertes.
hiScale	Realvariable zum Empfang des höchsten Wertes.

Beschreibung

Dieser Befehl wird verwendet, um die Konfiguration für den Kanal des Analogmodules zu lesen.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut

Beispiel einer ENetIO_AnaGetConfig Anweisung

```
Real gain, offset, loScale, hiScale

ENetIO_AnaGetConfig 8, gain, offset, loScale, hiScale
loScale = -5
hiScale = 5
ENetIO_AnaSetConfig 8, gain, offset, loScale, hiScale
```

ENetIO_AnaSetConfig Anweisung

Konfiguriert ein analoges Ethernet I/O Eingangs- oder Ausgangsmodul.

Syntax

ENetIO_AnaSetConfig *channel, gain, offset, loScale, hiScale*

Parameter

channel	Integer-Ausdruck oder I/O-Label, welches einen der analogen Ausgangsports repräsentiert.
gain	Real-Ausdruck, der die Eingangsverstärkung enthält. Setzen Sie ihn für die Vorgabeeinstellung auf 0. Normalerweise wird Gain für sehr präzise Spannungskalibrierung verwendet.
offset	Real-Ausdruck, der den Eingangs-Offset enthält. Setzen Sie ihn für die Vorgabeeinstellung auf 0. Normalerweise wird der Offset für sehr präzise Spannungskalibrierung verwendet.
loScale	Real-Ausdruck, der den niedrigsten auszugebenden Wert enthält.
hiScale	Real-Ausdruck, der den höchsten auszugebenden Wert enthält.

Beschreibung

Dieser Befehl wird verwendet, um die Konfiguration für den Kanal des Analogmodules einzustellen.

Zunächst sollten Sie die Konfigurationswerte mithilfe von ENetIO_AnaGetConfig lesen, danach die notwendigen Parameter ändern und dann ENetIO_SetConfig aufrufen.

Gain und Versatz

Normalerweise sollten Gain und Offset auf 0 eingestellt sein (werksseitige Vorgabeeinstellung). Wenn Sie Module präzise kalibrieren müssen, können Sie Offset und Gain unter Verwendung einer kalibrierten Quelle einstellen. Wenn Sie beispielsweise -50mv bis +50mv lesen müssen, das Modul jedoch diese Werte beim Vergleich mit einem Voltmeter nicht liest, stellen Sie Versatz und Gain ein, damit die Ablesungen korrekt sind.

Skalierung

Stellen Sie die *loScale* und *hiScale* Parameter ein, so dass sie den von Ihnen benötigten Einheiten entsprechen. Wenn Sie beispielsweise ein analoges Eingangsmodul verwenden, das -10 bis +10 Volt liest, können Sie *loScale* auf 0 setzen und *hiScale* auf 100, um den Prozentwert abzulesen.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut

Beispiel einer ENetIO_AnaSetConfig Anweisung

```
Real gain, offset, loScale, hiScale

ENetIO_AnaGetConfig 8, gain, offset, loScale, hiScale
loScale = -5
hiScale = 5
ENetIO_AnaSetConfig 8, gain, offset, loScale, hiScale
```

ENetIO_AnaOut Anweisung

Stellt das Ausgangslevel für einen analogen Ausgabekanal ein.

Syntax

ENetIO_AnaOut *channel, value*

Parameter

channel Integer-Ausdruck oder I/O-Label, welches einen der analogen Ausgangsports repräsentiert.

value Real-Ausdruck, der den Wert zur Einstellung des analogen Ausgabekanal repräsentiert.

Rückgabewerte

Beschreibung

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_AnaOut Funktion

```
ENetIO_AnaOut 8, 2.5
```

ENetIO_ClearLatches Anweisung

Löscht On- und Off-Latches für einen Ethernet I/O Digitaleingang.

Syntax

ENetIO_ClearLatches(*bitNumber*)

Parameter

bitNumber Integer-Ausdruck, welcher die Bitnummer repräsentiert.

Beschreibung

ENetIO_ClearLatches löscht sowohl das On-Latch als auch as Off-Latch für den angegebenen Punkt.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_ClearLatches Funktion

```
ENetIO_ClearLatches 1
```

ENetIO_In Funktion

F

Gibt den Status des angegebenen EtherNet I/O Eingangsports aus. Jeder Port enthält 4 EtherNet I/O Eingabekanäle.

Syntax

ENetIO_In(*portNumber*)

Parameter

portNumber Integer-Ausdruck, der einen der Eingangsports repräsentiert. Beachten Sie, dass jeder Port 4 Eingabekanäle hat.

Rückgabewerte

Gibt einen Integer-Wert zwischen 0 und 15 aus. Der Rückgabewert beträgt 4 Bits, wobei jedes Bit einem EtherNet I/O Eingabekanal entspricht.

Beschreibung

ENetIO_In gibt Ihnen die Möglichkeit, die Werte von 4 EtherNet I/O Eingabekanälen gleichzeitig zu betrachten.

Da 4 Kanäle zur gleichen Zeit geprüft werden, rangieren die Rückgabewerte zwischen 0 und 15. Bitte prüfen Sie die unten stehende Tabelle nach, um zu sehen, wie die Rückgabewerte den einzelnen EtherNet I/O Eingabekanälen entsprechen.

ENet Eingabekanal-Ergebnis

Rückgabewerte	3	2	1	0
1	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
5	Off (Aus)	On (Ein)	Off (Aus)	On (Ein)
15	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_In Funktion

```
Print ENetIO_In(1)
```

ENetIO_Off

S

Schaltet einen EtherNet I/O Ausgang aus.

Syntax

ENetIO_Off *bitNumber* [*, time*]

Parameter

bitNumber Integer-Ausdruck, der die auszuschaltende Bitnummer repräsentiert.

time Optionaler Parameter. Gibt ein Zeitintervall in Sekunden an, für das der Ausgang ausgeschaltet bleiben soll. Nachdem das Zeitintervall abgelaufen ist, wird der Ausgang wieder eingeschaltet. (Das minimale Zeitintervall beträgt 0.01 Sekunden)

Beschreibung

ENetIO_Off schaltet den angegebenen EtherNet I/O Ausgang aus (setzt ihn auf 0). Wenn der Zeitintervall-Parameter definiert ist, wird der durch *bitNumber* spezifizierte Ausgang ausgeschaltet, und nach Ablauf des Zeitintervalls wieder eingeschaltet. Wenn der Ausgang vor Auführung von ENetIO_Off bereits ausgeschaltet war, wird er nach Ablauf des Zeitintervalls wieder eingeschaltet.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_Off Anweisung

```
ENetIO_Off 1
```

ENetIO_On

S

Schaltet einen EtherNet I/O Ausgang ein.

Syntax

ENetIO_On *bitNumber* [*, time*]

Parameter

bitNumber Integer-Ausdruck, der die einzuschaltende Bitnummer repräsentiert.

time Optionaler Parameter. Gibt ein Zeitintervall in Sekunden an, für das der Ausgang eingeschaltet bleiben soll. Nachdem das Zeitintervall abgelaufen ist, wird der Ausgang wieder ausgeschaltet. (Das minimale Zeitintervall beträgt 0.01 Sekunden)

Beschreibung

ENetIO_On schaltet den angegebenen EtherNet I/O Ausgang ein (setzt ihn auf 1). Wenn der Zeitintervall-Parameter definiert ist, wird der durch *bitNumber* spezifizierte Ausgang eingeschaltet, und nach Ablauf des Zeitintervalls wieder ausgeschaltet.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_On Anweisung

```
ENetIO_On 1
```

ENetIO_Oport Funktion

F

Gibt den Status des angegebenen EtherNet I/O Ausgangs aus.

Syntax

ENetIO_Oport(*outnum***)**

Parameter

outnum Zahl, die einen der EtherNet I/O Ausgänge repräsentiert.

Rückgabewerte

Gibt den angegebenen Ausgangsstatus als entweder 0 oder 1 aus.

0: Off (Aus) Status

1: On (Ein) Status

Beschreibung

ENetIO_Oport gibt Ihnen die Möglichkeit, einen Status-Check für die EtherNet I/O Ausgänge durchzuführen.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_Oport Anweisung

```
Print ENetIO_Oport(5)
```

ENetIO_Out Anweisung

S

Setzt gleichzeitig 4 EtherNet I/O Ausgänge (ein Modul).

Syntax

ENetIO_Out *portNumber, outData*

Parameter

portNumber **Integer-Ausdruck, der eine der Ausgangsgruppen repräsentiert (jede Gruppe enthält 4 Ausgänge) aus denen die EtherNet I/O Ausgänge bestehen**.

outData Integer-Ausdruck zwischen 0 und 15, der das Ausgangsmuster für die mit der Anweisung *portNumber* ausgewählte Ausgangsgruppe repräsentiert. Wenn er in hexadezimaler Form dargestellt wird, wird der Bereich von &H0 bis &HF abgedeckt.

Beschreibung

ENetIO_Out setzt unter Nutzung der vom Anwender angegebenen Kombination aus *portNumber* und *outData* Werten gleichzeitig 4 EtherNet I/O Ausgänge, um festzusetzen, welche Ausgänge eingestellt werden.

Sobald eine Portnummer ausgewählt wurde (d.h. es wurde eine Gruppe von vier Ausgängen ausgewählt), muss ein spezifisches Ausgangsmuster definiert werden. Dafür wird der Parameter *outData* verwendet. Der *outData* Parameter kann einen Wert zwischen 0 und 15 haben und kann in hexadezimalem oder Integer Format repräsentiert werden. (d.h. &H0-&HF oder 0-15)

Die folgende Tabelle zeigt einige der möglichen Ausgangskombinationen und ihre zugehörigen *outData*-Werte.

Ausgangs-Einstellungen

outData Wert	3	2	1	0
01	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
02	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)
03	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
08	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)
09	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)
15	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Sw

ENetIO_Out Beispiel

```
ENetIO_Out 0,&H0F
```

ENetIO_Sw Funktion

F

Gibt den Status des ausgewählten EtherNet I/O Eingangsports aus oder zeigt ihn an.

Syntax

ENetIO_Sw(*bitNumber***)**

Parameter

bitNumber Zahl, die einen der EtherNet I/O Eingänge repräsentiert.

Rückgabewerte

Gibt eine 1 aus, wenn der angegebene EtherNet I/O Eingang eingeschaltet (On) ist und eine 0, wenn der angegebene EtherNet I/O Eingang ausgeschaltet (Off) ist.

Beschreibung

ENetIO_Sw gibt Ihnen die Möglichkeit, einen Status-Check für die EtherNet I/O Eingänge durchzuführen. Der EtherNet I/O Eingang, der mit dem **ENetIO_Sw**-Befehl überprüft wurde, hat 2 Statûs (1 oder 0). Diese zeigen an, ob der Eingang ein- oder ausgeschaltet ist.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out

Beispiel einer ENetIO_Sw Funktion

```
Print ENetIO_Sw(5)
```

ENetIO_SwLatch Funktion

F

Liest On-Latch und Off-Latch des EtherNet I/O Digitaleinganges.

Syntax

ENetIO_SwLatch(*bitNumber, latchType*)

Parameter

bitNumber Integer-Ausdruck oder I/O-Label, welches einen der EtherNet I/O Digitaleingänge repräsentiert.

latchType **On | Off**

Beschreibung

Jeder digitale Eingang hat zwei Latches. Das On-Latch wird gesetzt für den Fall, dass der Eingang jemals von Aus nach Ein umschaltet. Das Off-Latch wird gesetzt für den Fall, dass der Eingang jemals von Ein nach Aus umschaltet. **ENetIO_SwLatch** wird verwendet, um den Status eines Latches abzulesen.

ENetIO_ClearLatches wird verwendet, um beide Latches zu löschen.

Hinweis

Dieser Befehl funktioniert nur dann, wenn die EtherNet I/O Option installiert ist.

Verwandte Befehle

ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

Beispiel einer ENetIO_SwLatch Funktion

```
Boolean latchStatus

latchStatus = ENetIO_SwLatch(0, On)
```

Eof Funktion

F

Gibt den Status des Dateiendes aus.

Syntax
Eof (*fileNumber)*
Parameter
fileNumber Integer-Ausdruck, der die zu prüfende Zahl repräsentiert.
Rückgabewerte
WAHR, wenn sich der Dateizeiger am Ende der Datei befindet, ansonsten FALSCH.

Verwandte Befehle
Lof

Eof Beispiel

```
ROpen "test.dat" As #30
Do While Not Eof(30)
    Line Input #30, tLine$
    Print tLine$
Loop
Close #30
```

EPrint Anweisung

> S

Der **EPrint** (end print = Druck beenden) Befehl sendet Daten an den PC-Drucker, nachdem der LPrint Befehl verwendet wurde.

Syntax

EPrint

Beschreibung

EPrint wird nach Ausführung der letzten **LPrint** Anweisung verwendet.

Verwandte Befehle

LPrint, Print

EPrint Beispiel

```
Function dataPrint
    Integer i

    For i = 1 to 10
        LPrint testDat$(i)
    Next i
    EPrint   ' Schickt die Daten an den Drucker des PCs
Fend
```

Era Funktion

F

Gibt die Achsnummer aus, für die ein Fehler aufgetreten ist.

Syntax

Era(*taskNum*)

Parameter

taskNum Integer-Ausdruck, der eine Tasknummer von 0 bis 32 repräsentiert. 0 gibt den aktuellen Task an.

Rückgabewerte

Die Achsnummer, die den Fehler im Bereich 0-4 verursacht hat, wie im folgenden beschrieben:

0 – Der aktuelle Fehler wurde nicht durch eine Servachse verursacht.
1 - Der aktuelle Fehler wurde durch Achsnummer 1 (T1) verursacht
2 - Der aktuelle Fehler wurde durch Achsnummer 2 (T2) verursacht
3 - Der aktuelle Fehler wurde durch Achsnummer 3 (Z) verursacht
4 - Der aktuelle Fehler wurde durch Achsnummer 4 (U rotationsfähige Ausrichtungsachse) verursacht

Beschreibung

Era wird verwendet, wenn ein Fehler auftritt, um festzustellen, ob der Fehler von einem der vier Servoachsen verursacht wurde und um die Nummer der Achse auszugeben, die den Fehler verursacht hat. Wenn der aktuelle Fehler nicht von einer Achse verursacht wurde, gibt die **Era**-Funktion eine 0 (Null) aus.

Verwandte Befehle

EClr, Erl, Err, Errhist, ErrMsg$, Ert, OnErr, Trap

Beispiel einer Era Funktion

Das folgende Beispiel zeigt ein einfaches Programm, das die Era-Funktion verwendet, um festzustellen, in welchem Task und in welchem Zusammenhang der Fehler auftrat; Erl: wo der Fehler auftrat; Err: welcher Fehler auftrat; Era: ob der Fehler von einer Achse verursacht wurde.

```
Trap Error Call ER_PRINT     'Stellt ein, wann ein Fehler auftritt
  .
  .
Function ER_PRINT            'Fehlerbehandlungsroutine

    Integer errTask

    errTask = t
    Print "Task number at which error occurred is ", errTask
    If Era(errTask) Then
        Print "Joint which caused error is ", Era(errTask)
    EndIf
    Print "The Error code is ", Err(errTask)
    Print "The line where the error occurred is Line ", Erl(errTask)
Fend
```

Erase Anweisung

S

Löscht alle Elemente einer Matrix oder einer Variable.

Syntax

Erase { *varName* | **All** }

Parameter

varName	Name einer zu löschenden Matrix oder Variable.
All	Löscht alle Matrizen oder Variablen im aktuellen Projekt.

Beschreibung

Erase kann zum Löschen jeglicher Variable verwendet werden. Nummervariablen werden auf 0 gesetzt und Zeichenkettenvariablen werden auf leere Zeichenketten gesetzt.

Erase All wird normalerweise verwendet, um alle Variablen zurückzusetzen, nachdem ein Chain-Befehl ausgeführt worden ist. Wenn ein Programm jedoch vom Run- oder vom Operator-Fenster aus gestartet wird, werden alle Variablen automatisch gelöscht.

Verwandte Befehle

Redim, UBound

Beispiel einer Erase Anweisung

```
Integer i, a(10)

For i= 1 To 10
    a(i) = i
Next i

' Löscht alle Elemente von a
Erase a
```

EResume Anweisung

S

Setzt die Ausführung nach Beendigung einer Fehlerbehandlungsroutine fort.

Syntax

EResume [**Next**]

Beschreibung

EResume

Wenn der Fehler in derselben Prozedur auftrat, wie die Fehlerbehandlungsroutine, wird die Ausführung mit der Anweisung, die den Fehler verursacht hat, fortgesetzt Wenn der Fehler in einer aufgerufenen Prozedur auftrat, wird die Ausführung mit einer Anweisung fortgesetzt. Als Anweisung wird diejenige verwendet, welche zuletzt aus der Prozedur, die eine Fehlerbearbeitungsroutine enthält, aufgerufen wurde.

EResume Next

Wenn der Fehler in derselben Prozedur auftrat, wie die Fehlerbehandlungsroutine, wird die Ausführung mit der Anweisung fortgesetzt, welche direkt auf diejenige folgt, welche den Fehler verursacht hat. Wenn der Fehler in einer aufgerufenen Prozedur auftrat, wird die Ausführung mit einer Anweisung fortgesetzt. Als Anweisung wird diejenige verwendet, die direkt auf jene folgt, welche zuletzt aus der Prozedur, die eine Fehlerbearbeitungsroutine enthält, aufgerufen wurde.

Verwandte Befehle

OnErr

Beispiel einer EResume Anweisung

```
Function LogData(data$ As String)

   OnErr GoTo LogDataErr
   Integer answer
   String msg$

   WOpen "a:\test.txt" As #30
   Print #30, data$
   Close #30

   Exit Function

LogDataErr:
   msg$ = "Cannot open file on floppy." + CRLF
   msg$ = msg$ + "Please insert floppy in drive."
   MsgBox msg$, MB_OK, "Error", answer
   EResume
Fend
```

Erl Funktion

F

Gibt die Zeilennummer aus, in welcher der Fehler aufgetreten ist.

Syntax

Erl(*taskNumber*)

Parameter

taskNumber Integer-Ausdruck, der eine Tasknummer von 0 bis 32 repräsentiert. 0 gibt den aktuellen Task an.

Rückgabewerte

Zeilennummer, in welcher der letzte Fehler auftrat.

Beschreibung

Erl wir zusammen mit OnErr verwendet. Wenn ein Fehler in einem Programm innerhalb von OnErr...Return auftritt, ist es mitunter schwierig, die Fehlerquelle festzustellen. Erl gibt die Zeilennummer aus, in welcher der Fehler aufgetreten ist. Wird **Erl** in Kombination mit Err, Ert und Era verwendet, erhält der Anwender wesentlich mehr Informationen über den aufgetretenen Fehler.

Einschränkungen

Denken Sie daran, Erl auszuführen, bevor Sie EClr verwenden:

Da der im Fehlerbehandlungsprogramm interne EClr-Befehl die Fehlerzeilennummer auf 0 zurücksetzt, sollten Sie sichergehen, dass Sie **Erl** vor EClr ausführen. Anderenfalls werden Sie nicht feststellen können, in welcher Zeilennummer der Fehler auftrat.

Verwandte Befehle

EClr, Era, Err, Errhist, ErrMsg$, Ert, OnErr, Trap

Beispiel einer Erl Funktion

Das folgende Beispiel zeigt ein einfaches Programm, das die Ert-Funktion verwendet, um festzustellen, in welchem Task und in welchem Zusammenhang der Fehler auftrat; Erl: wo der Fehler auftrat; Err: welcher Fehler auftrat; Era: ob der Fehler von einer Achse verursacht wurde.

```
Trap Error Call ER_PRINT     'Stellt ein, wann ein Fehler auftritt
  .
  .
Function ER_PRINT            'Fehlerbehandlungsroutine
    Print "Task number at which error occurred is ", Ert
    If Era(0) Then
        Print "Joint which caused error is ", Era(0)
    EndIf
    Print "The Error code is ", Err(0)
    Print "The line where the error occurred is Line ", Erl(0)
Fend
```

Err Funktion

F

Gibt den zuletzt aufgetretenen Fehlerstatus aus.

Syntax

Err [(*taskNumber*)]

Parameter

taskNumber	Optional. Integer-Ausdruck, der eine Tasknummer von 0 bis 32 repräsentiert. 0 gibt den aktuellen Task an.

Rückgabewerte

Gibt einen numerischen Fehlercode in Integer Form aus.

Beschreibung

Err gestattet es dem Anwender, den aktuellen Fehlercode zu lesen. Zusammen mit den Fehlerbehandlungsfähigkeiten von SPEL versetzt dies den Anwender in die Lage festzustellen, welcher Fehler aufgetreten ist und entsprechend zu reagieren. **Err** wir zusammen mit OnErr verwendet. Wenn ein Fehler in einem Programm innerhalb von OnErr...Return auftritt, kann es schwierig sein, die Fehlernummer festzustellen. Durch Verwnedung der Err-Funktionen wird dieser Prozess jedoch leicht gemacht.

Einschränkungen

Denken Sie daran, dass alle Fehler mit der EClr-Anweisung gelöscht werden müssen:

Wenn der Anwender in der Fehlerbehandlungsroutine jedoch keine EClr Anweisung verwendet, um den Fehler zu löschen, kann das System in einer Endlosschleife stecken bleiben. Wenn der Fehler nicht mithilfe der Fehlerbehandlungsroutine zurückgesetzt wird, gibt das System auf den OnErr-Befehl hin die Steuerung sofort nach seiner Rückkehr zur Hauptroutine an die Fehlerbehandlungsroutine zurück. Der Grund dafür ist, dass das System davon ausgeht, es sei erneut ein Fehler aufgetreten, während es tatsächlich nur auf den vorangegangenen Fehler reagiert (der nicht gelöscht wurde). Da die Fehlerbehandlungsroutine den Fehler nicht zurücksetzt, tritt eine Endlosschleife auf. (Seien Sie daher vorsichtig!)

Verwandte Befehle

EClr, Era, Erl, ErrHist, ErrMsg$, Ert, OnErr, Return, Trap

Err Beispiel

Das folgende Beispiel zeigt ein einfaches Dienstprogramm, das prüft, ob ob die Punkte P0-P399 existieren. Wenn der Punkt nicht existiert, erscheint eine Nachricht auf dem Bildschirm, um den Anwender diesen Umstand wissen zu lassen. Das Programm verwendet den CX-Befehl, um jeden einzelnen Punkt dahin gehend zu testen, ob er definiert wurde, oder nicht. Wenn ein Punkt nicht definiert ist, wird die Steuerung an die Fehlerbehandlungsroutine übergeben und auf dem Bildschirm erscheint eine Nachricht, die dem Anwender mitteilt, welcher Punkt nicht definiert war. Beachten Sie, dass der Fehlercode vom System in *errnum* Variable gespeichert wird, bevor Sie **ECIr** aufrufen.

```
Function eclrtest
   Integer i, errnum
   Real temp

   OnErr Goto eHandle
   For i = 0 To 399
     temp = CX(P(i))
   Next i
   End
'
'
'*****************************************
'* Error Handler                          *
'*****************************************
eHandle:
   errnum = Err(0); EClr
   'Überprüft, ob ein undefinierter Punkt verwendet wird
   If errnum = 78 Then
      Print "Point number P", i, " is undefined!"
   Else
      Print "ERROR: Error number ", errnum, " Occurred."
   End If
   EResume Next
Fend
```

ErrHist Befehl

>

Zeigt ein Protokoll unlängst aufgetretener Fehler an.

Syntax

ErrHist [{**CLEAR** | *filter*}]

Parameter

CLEAR	Optional. Löscht das gesamte Protokoll.
filter	Optional. Filterabfrage nach einer Protokollauflistung. Sie können jegliche Zeichenkette angeben. Jede Protokollzeile, welche die Filter-Zeichenkette enthält, wird angezeigt.

Beschreibung

Anzeige der zuletzt aufgetretenen Fehler. Die folgenden Punkte werden angezeigt:

- Datum und Uhrzeit des Auftretens des Fehlers
- EPSON RC+ Version
- Fehlertyp: System, RunTime, Monitor
- Fehlernummer
- Fehlermeldung

Wenn sie zutreffen, werden auch die folgenden Punkte angezeigt:

- Funktionsname
- Tasknummer
- Zeilennummer
- Roboternummer
- Achsnummer

Hinweise

Fehler, die von einer Fehlerbehandlungsroutine aufgefangen werden, werden nicht aufgezeichnet

- Wenn OnErr verwendet wird, werden jegliche Fehler, die von OnErr aufgefangen werden, nicht aufgezeichnet.

Verwandte Befehle

EClr, Era, Erl, Err, Errhist, ErrMsg$, Ert, OnErr, Trap

Errhist Beispiel

```
> errhist
11/15/1999 08:17:37, v3.0.0, RunTime, main, 150, Command not allowed with
motors off., Task 1, Line 3, Robot 1
11/15/1999 08:18:24, v3.0.0, RunTime, main, 125, The arm reached the limit
of motion range., Task 1, Line 5, Robot 1, Joint 4
```

Zeigt mithilfe des optionalen Filter-Parameters alle Fehler an, die das Wort “joint” (“Achse”) enthalten.

```
> errhist joint
11/15/1999 08:18:24, v3.0.0, RunTime, main, 125, The arm reached the limit
of motion range., Task 1, Line 5, Robot 1, Joint 4
>
```

ErrMsg$ Funktion

F

Gibt die Fehlermeldung aus, die der angegebenen Fehlernummer entspricht.

Syntax

ErrMsg$(*errNumber*)

Parameter

errNumber Integer-Ausdruck, der die Fehlernummer enthält, für welche die Nachricht ausgegeben wird.

Rückgabewerte

Gibt die Fehlermeldung aus, die in der Fehlercode-Tabelle beschrieben ist.

Verwandte Befehle

EClr, Era, Erl, Err, Errhist, Ert, OnErr, Trap

ErrMsg$ Beispiel

Das folgende Beispiel zeigt ein einfaches Programm, das die Ert-Funktion verwendet, um festzustellen, in welchem Task und in welchem Zusammenhang der Fehler auftrat; Erl: wo der Fehler auftrat; Err: welcher Fehler auftrat; Era: ob der Fehler von einer Achse verursacht wurde.

```
   Trap Error Call ER_PRINT   'Einstellung für Fehlerfall
  .
  .
Function ER_PRINT       'Fehlerbehandlungsroutine
     Print "Task number at which error occurred is ", Ert
     If Era Then Print "Joint which caused error is ", Era
     Print "The Error code is ", Err
     Print "The Error Message is ", ErrMsg$(Err)
     Print "The line where the error occurred is Line ", Erl
     End
Fend
```

Error Anweisung

S

Simuliert das Auftreten eines Fehlers.

Syntax
Error *errorNumber*

Parameter

errorNumber	Integer-Ausdruck, der eine gültige Fehlernummer repräsentiert. Die Anwender-Fehlernummern beginnen bei 30000.

Beschreibung
Verwenden Sie die Error-Anweisung, um system- oder benutzerdefinierte Fehler zu generieren.

Verwandte Befehle
EClr, Era, Erl, Err, ErrHist, OnErr

Beispiel einer Error Anweisung

```
#define ER_VAC 30000

If Sw(vacuum) = Off Then
    Error ER_VAC
EndIf
```

Ert Funktion

> F

Gibt die Tasknummer aus, in der ein Fehler aufgetreten ist.

Syntax

Ert

Rückgabewerte

Die Tasknummer, in welcher der Fehler aufgetreten ist.

Beschreibung

Wenn ein Fehler auftritt wird **Ert** verwendet um festzustellen, in welchem Task der Fehler aufgetreten ist. Die ausgegebene Zahl wird zwischen 1 und 32 liegen.

Verwandte Befehle

EClr, Era, Erl, Err, ErrHist, ErrMsg$, OnErr, Trap

Beispiel einer Ert Funktion

Das folgende Beispiel zeigt ein einfaches Programm, das die Ert-Funktion verwendet, um festzustellen, in welchem Task und in welchem Zusammenhang der Fehler auftrat; Erl: wo der Fehler auftrat; Err: welcher Fehler auftrat; Era: ob der Fehler von einer Achse verursacht wurde.

```
Trap Error Call ER_PRINT      'Einstellung für Fehlerfall
  .
  .
Function ER_PRINT       'Fehlerbehandlungsroutine
    Print "Task number at which error occurred is ", Ert
    If Era(0) Then
        Print "Joint which caused error is ", Era(0)
    EndIf
    Print "The Error code is ", Err(0)
    Print "The Error Message is ", ErrMsg$(Err(0))
    Print "The line where the error occurred is Line ", Erl(0)
    End
Fend
```

EStopOn Funktion

F

Gibt den aktuellen “Emergency Stop” (Not-Halt) Status aus.

Syntax

EStopOn

Rückgabewerte

WAHR, wenn der Emergency Stop Kreis unterbrochen wurde (ON), sonst FALSCH.

Beschreibung

EStopOn gibt WAHR aus, nachdem der Emergency Stop Kreis ausgelöst wurde. Selbst nach Schließen des Kreises (was normalerweise durch herausziehen des Not-Halt-Schalters geschieht), gibt **EStopOn** TRUE aus. Sie müssen Reset oder Reset Emergency ausführen, um den Zustand zu löschen.

EStopOn entspricht ((Stat(0) And &H100000) <> 0).

Verwandte Befehle

PauseOn, SafetyOn, Stat

Beipiel einer EStopOn Funktion

```
Function main

   Trap Emergency Call eTrap

   Do
      Print "Program running..."
      Wait 1
   Loop

Fend

Function eTrap

   String msg$
   Integer ans
   msg$ = "Estop Is on.  Please clear it and Retry"

   Do While EStopOn
      MsgBox msg$, MB_RETRYCANCEL, "Clear E-stop and click Retry", ans
      If ans = IDRETRY Then
         Reset
      Else
         Exit Function
      EndIf
      Wait 1
   Loop

   Restart

Fend
```

Eval Funktion

F

Führt eine Online-Fenster-Anweisung aus und gibt den Fehlerstatus aus.

Syntax

Eval(*command* [, *reply$*])

Parameter

command	Ein Zeichenkettenausdruck, der einen Online-Modusbefehl enthält.
reply$	Optional. Eine Zeichenkettenvariable, die die Antwort des Befehls beinhaltet.

Rückgabewerte

Der Fehlercode, der vom Ausführen des Befehls ausgegeben wird.

Beschreibung

Eval wird verwendet, um Online-Befehle von einem anderen Gerät aus auszuführen. Beispielsweise können Sie Befehle über RS-232 oder TCP/IP eingeben und sie dann ausführen.

Verwenden Sie den *reply$* Parameter um die Anwort von dem Befehl auszugeben. Wenn z.B.der Befehl "Print Sw(1)" lautete, würde *reply$* eine 1 oder eine 0 enthalten.

Verwandte Befehle

Error Codes

Beispiel einer Eval Funktion

Dieses Beispiel zeigt, wie man einen Befehl ausführt, der über RS-232 gelesen wird. Nachdem der Befehl ausgeführt wurde, wird der Fehlercode an den Host zurückgegeben. Der Host könnte beispielsweise einen Befehl wie "motor on" senden.

```
Integer errCode
String cmd$

OpenCom #1
Do
  Line Input #1, cmd$
  errCode = Eval(cmd$)
  Print #1, errCode
Loop
```

Exit Anweisung

S

Verlässt ein Schleifenkonstrukt oder eine Funktion.

Syntax

Exit { **Do | For | Function** }

Beschreibung

Die Syntax der Exit Anweisung hat die folgende Form:

Anweisung	Beschreibung
Exit Do	Stellt eine Möglichkeit dar, eine Do...Loop Anweisung zu verlassen. Es kann nur innerhalb einer Do...Loop Anweisung verwendet werden. **Exit Do** überträgt die Steuerung an die Anweisung, die auf die Loop-Anweisung folgt. Wenn **Exit Do** innerhalb verschachtelter **Do...Loop** Anweisungen verwendet wird, überträgt **Exit Do** die Steuerung an den Loop der sich eine Verschachtelungsebene über dem Loop befindet, in dem **Exit Do** auftritt.
Exit For	Stellt eine Möglichkeit dar, eine For-Schleife zu verlassen. Es kann nur innerhalb einer For…Next Schleife verwendet werden. **Exit For** überträgt die Steuerung an die Anweisung, welche auf die Next-Anweisung folgt. Wenn **Exit For** innerhalb verschachtelter **For** Schleifen verwendet wird, überträgt **Exit For** die Steuerung an die Schleife der sich eine Verschachtelungsebene über der Schleife befindet, in dem **Exit For** auftritt.
Exit Funktion	Verlässt sofort die Funktionsprozedur in der es erscheint. Die Ausführung fährt mit der Anweisung fort, welche auf diejenige folgt, die die Funktion aufgerufen hat.

Verwandte Befehle

Do...Loop, For...Next, Function...Fend

Beipiel einer Exit Anweisung

```
For i = 1 To 10
    If Sw(1) = On Then
        Exit For
    EndIf
    Jump P(i)
Next i
```

FileDateTime$ Funktion

F

Gibt Dateidatum und –zeit aus.

Syntax

FileDateTime$(*fileName*)

Parameter

fileName	Ein Zeichenkettenausdruck, der den zu überprüfenden Dateinamen enthält. Laufwerk und Pfad können ebenfalls enthalten sein.

Rückgabewerte

Gibt eine Zeichenkette aus, die Datum und Uhrzeit der letzten Änderung in folgendem Format enthält:

m/d/yyyy hh:mm:ss

Verwandte Befehle

FileExists, FileLen

Beispiel einer FileDateTime$ Funktion

```
Print FileDateTime$("c:\data\myfile.txt")
```

FileExists Funktion

F

Überprüft die Existenz einer Datei.

Syntax

FileExists(*fileName*)

Parameter

fileName	Ein Zeichenkettenausdruck, der den zu überprüfenden Dateinamen enthält. Laufwerk und Pfad können ebenfalls enthalten sein.

Rückgabewerte

WAHR, wenn die Datei existiert, FALSCH, wenn sie nicht existiert.

Verwandte Befehle

FolderExists

Beispiel einer FileExists Funktion

```
If FileExists("d:\data\mydata.dat") Then
    Kill "d:\data\mydata.dat"
EndIf
>
```

FileLen Funktion

F

Gibt die Länge einer Datei aus.

Syntax

FileLen(*fileName*)

Parameter

fileName	Ein Zeichenkettenausdruck, der den zu überprüfenden Dateinamen enthält. Laufwerk und Pfad können ebenfalls enthalten sein.

Rückgabewerte

Gibt die Anzahl von Bytes in der Datei aus.

Verwandte Befehle

FileDateTime$, FileExists

Beispiel einer FileExists Funktion

```
Print FileLen("c:\data\mydata.txt")
>
```

Find Anweisung

S

Gibt die Bedingung an, unter der Koordinaten während einer Bewegung gespeichert werden oder zeigt sie an.

Syntax

Find [*condition*]

Parameter

condition Konditionaler Ausdruck der veranlasst, dass die Koordinaten gespeichert werden.

Verwandte Befehle

FindPos, Go, Jump, PosFound

Beispiel einer Find Anweisung

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
End If
```

FindPos Funktion

F

Gibt einen Roboterpunkt aus, der während einer Bewegung durch die Find-Anweisung gespeichert wurde.

Syntax

FindPos

Rückgabewerte

Ein Roboterpunkt, der während einer Bewegung durch die Find-Anweisung gespeichert wurde.

Verwandte Befehle

Find, Go, Jump, PosFound, CurPos, InPos

Beispiel einer FindPos Funktion

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
End If
```

Fine Anweisung

> S

Definiert die Positionierungsgenauigkeit für Zielpunkte und zeigt sie an.

Syntax

Fine [*j1MaxErr, j2MaxErr, j3MaxErr, j4MaxErr*]

Parameter

j1MaxErr Integer-Ausdruck im Bereich von (0-32767) der den zulässigen Positionierungsfehler für die erste Achse repräsentiert. (SCARA: T1 Achse, kartesischer Roboter: X Achse).

j2MaxErr Integer-Ausdruck im Bereich von (0-32767) der den zulässigen Positionierungsfehler für die zweite Achse repräsentiert. (SCARA: T2 Achse, kartesischer Roboter: Y Achse).

j3MaxErr Integer-Ausdruck im Bereich von (0-32767) der den zulässigen Positionierungsfehler für die dritte Achse repräsentiert. (Z Achse).

j4MaxErr Integer-Ausdruck im Bereich von (0-32767) der den zulässigen Positionierungsfehler für die vierte Achse repräsentiert. (U Achse).

Rückgabewerte

Wenn **Fine** ohne Parameter verwendet wird, werden die aktuellen Fine-Werte für jede der vier Achsen angezeigt.

Beschreibung

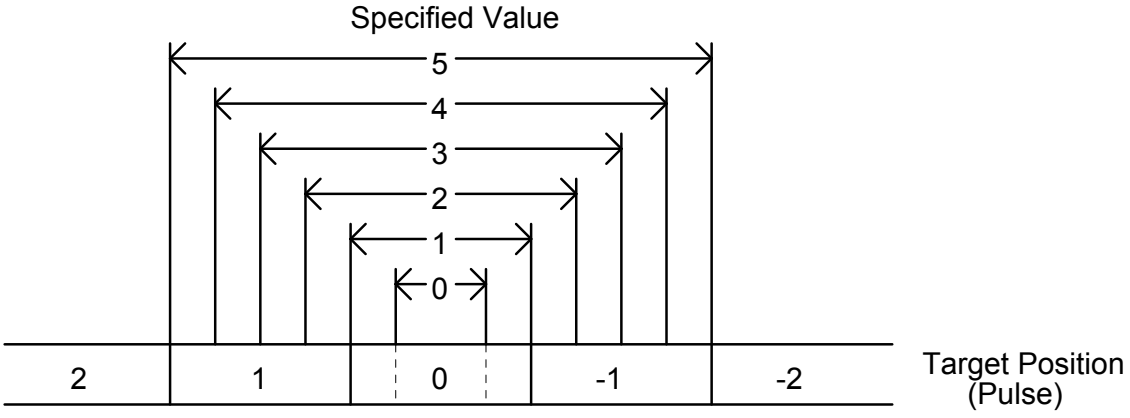
Fine definiert für jede Achse den zulässigen Positionierungsfehler, mit dem die Vollendung beliebiger Bewegungen registriert wird.

Die Prüfung hinsichtlich der Bewegungsvollendung beginnt, nachdem die CPU in der Drive Unit den Zielpositions-Pulswert vollständig an das Servo-System gesandt hat. Aufgrund der Servoverzögerung wird der Roboter die Zielposition noch nicht erreicht haben. Diese Prüfung wird im Abstand von wenigen Millisekunden wiederholt, bis jede Achse in der angegebenen Bereichseinstellung angekommen ist. Die Positionierung wird als abgeschlossen betrachtet, wenn alle Achsen innerhalb dieser angegebenen Bereiche angekommen sind. Sobald die Positionierung abgeschlossen ist, geht die Programmsteuerung auf den nächsten Befehl über.

Wenn mit dem Fine-Befehl vergleichsweise große Bereiche verwendet werden, wird die Positionierung relativ früh in der Bewegung bestätigt, was eine eher grobe Zielpositionierung zur Folge hat.

Die im Folgendenden dargestellte Formel und Abbildung verdeutlicht die Beziehung zwischen den definierten Werten und der entsprechenden Positionierungsgenauigkeit.

Positionierungsgenauigkeit = +/- (n+1) * 1/4 Pulse (wobei es sich bei n um den unten angebenen Wert handelt)



Tipps

Zykluszeiten und der Fine-Befehl

Der **Fine** Wert hat keine Auswirkungen auf die Steuerung der Beschleunigung oder Verzögerung des Manipulatorarms. Kleinere **Fine** Werte können das System jedoch veranlassen, langsamer zu laufen, da es das Servosystem zusätzliche Zeit kosten kann (einige Millisekunden) um in einen zulässigen Positionsbereich zu gelangen. Sobald der Arm den zulässigen Positionsbereich erreicht hat (definiert durch den **Fine** Befehl), führt die CPU den nächsten Anwender-Befehl aus. (Bedenken Sie, dass alle aktivierten Achsen in Position sein müssen, bevor die CPU den nächsten Anwender-Befehl ausführen kann.)

Da die Servoverzögerung normalerweise weniger als einige tausend Pulse beträgt, wird eine Bereichseinstellung von ein paar tausend Pulsen oder mehr keinen vorteilhaften Positionierungseffekt bringen.

Initialisierung von Fine (durch die Befehle Motor On, SLock, SFree und Verinit)

Jedes Mal, wenn die folgenden Werte verwendet werden, wird der Fine Wert auf 10 gesetzt (initialisiert) (der **Fine** Vorgabewert): bei den Befehlen SLock, SFree, Motor und Verinit.

Stellen Sie sicher, dass Sie die Fine-Werte neu einstellen, nachdem einer der o.g. Befehle ausgeführt wurde.

Potentielle Fehler

Wenn die **Fine** Positionierung nicht innerhalb von 2 Sekunden abgeschlossen ist, tritt Fehler 151 auf.

Verwandte Befehle

Accel, AccelS, Arc, Go, Jump, Move, Speed, SpeedS, Pulse

Beispiel einer Fine Anweisung

Das folgende Beispiel zeigt eine einfache PTP-Bewegung zwischen den Punkten P0 und P1 und die darauf folgende geradlinige Rückkehr zum Punkt P0. Später im Programm bewegt sich der Arm zu Punkt P2, bis Eingang #2 einschaltet. Wenn Eingang #2 während der Bewegung einschaltet, dann verzögert sich der Arm bis zum vollständigen Stopp, bevor er Punkt P2 erreicht und der nächste Programmbefehl wird ausgeführt.

```
Function finetest
    Home
    Fine 5, 5, 5, 5           'Reduziert die Präzision auf +/- 1 Pulse
    Go P1
    Go P2 Till Sw(2) = On
    If Sw(2) = On Then
       Print "Input #2 came on during the move and"
       Print "the robot stopped prior to arriving on"
       Print "point P2."
    Else
       Print "The move to P2 completed successfully."
       Print "Input #2 never came on during the move."
    End If
Fend
```

< Online-Fenster -Beispiel>

```
> Fine 50, 50, 50, 50
>
> Fine
10, 10, 10, 10
```

Fine Funktion

F

Gibt die **Fine** Einstellung für eine angegebene Achse aus.

Syntax

Fine(*joint*)

Parameter

joint Integer-Ausdruck, der die Achsnummer repräsentiert, für welche die Fine-Einstellungen ausgegeben werden sollen.

Rückgabewerte

Real-Wert

Verwandte Befehle

Accel, AccelS, Arc, Go, Jump, Move, Speed, SpeedS, Pulse

Beispiel einer Fine Funktion

Dieses Beispiel benutzt die **Fine**-Funktion in einem Programm:

```
Function finetst
    Integer a
    a = Fine(1)
Fend
```

Fix Funktion

F

Gibt den Integer Anteil einer Real-Zahl aus.

Syntax
Fix(*number*)
Parameter
number Real-Ausdruck, der die zu fixierende Zahl enthält.

Rückgabewerte
Ein Integer-Wert, der den Integer Anteil einer Real-Zahl enthält.

Verwandte Befehle
Int

Beispiel einer Fix Funktion

```
>print Fix(1.123)
 1
>
```

FmtStr Anweisung

S

Formatiert einen numerischen oder einen Datum/Zeit Ausdruck.

Syntax

FmtStr *expression, strFormat, strOut$*

Parameter

expression	Zu formatierender Ausdruck.
strFormat	Formatspezifizierungs-Zeichenkette.
strOut$	Ausgangs-Zeichenkettenvariable.

Beschreibung

FmtStr wird verwendet, um einen numerischen Ausdruck in eine Zeichenkette umzuwandeln.

Spezifikationssymbole für das numerische Format

Zeichen	Beschreibung
Keins	Anzeige der Zahl ohne Formatierung.
(0)	Zahlzeichen-Platzhalter. Anzeige eines Zahlzeichens oder einer Null. Wenn der Ausdruck ein Zahlzeichen an der Stelle hat, an welcher in der Formatzeichenkette eine Null (0) erscheint, so wird es angezeigt. Ansonsten wird an dieser Stelle eine Null angezeigt. Wenn eine Zahl weniger Zahlzeichen hat (egal auf welcher Seite der Dezimalen), als der Formatausdruck Nullen aufweist, werden die vorangehenden oder die anhängenden Nullen angezeigt. Wenn die Zahl mehr Zahlzeichen zur Rechten der Dezimaltrennung hat, als Nullen zur rechten Seite der Dezimaltrennung im Formatausdruck vorhanden sind, wird diese Zahl auf so viele Dezimalstellen gerundet, wie Nullen vorhanden sind. Wenn die Zahl mehr Zahlzeichen zur Linken der Dezimaltrennung hat, als Nullen zur linken Seite der Dezimaltrennung im Formatausdruck vorhanden sind, werden die überzähligen Zahlzeichen ohne Änderung angezeigt.
(#)	Zahlzeichen-Platzhalter. Zeigt ein Zahlzeichen oder gar nichts an. Wenn der Ausdruck ein Zahlzeichen an der Stelle hat, an welcher in der Formatzeichenkette ein #-Zeichen erscheint, so wird es angezeigt. Ansonsten wird an dieser Stelle gar nichts angezeigt. Dieses Symbol arbeitet wie der 0 Zahlzeichen-Platzhalter, mit der Ausnahme, dass die vorangehenden und die anhängenden Nullen nicht angezeigt werden, wenn die Zahl ebenso viele oder weniger Zahlzeichen, als #-Zeichen auf beiden Seiten der Dezimaltrennung im Formatausdruck.
(.)	Dezimaler Platzhalter. In einigen Darstellungen wird ein Komma zur Tennung von Dezimalstellen verwendet. Der dezimale Platzhalter bestimmt, wie viele Zahlzeichen zur Linken und Rechten der Dezimalstellen-Trennung angezeigt werden. Wenn der Formatausdruck ausschließlich Nummernzeichen zur Linken dieses Symbols beinhaltet, beginnen Zahlen kleiner als 1 mit einer Dezimalstellen-Trennung. Um eine vorangehende Null mit Nachkommastellen anzuzeigen, verwenden Sie 0 als ersten Zahlzeichen-Platzhalter zur Linken der Dezimalstellen-Trennung. Das Zeichen, welches tatsächlich als dezimaler Platzhalter in den formatierten Ausgabedaten verwendet wird, hängt ab von dem Nummern-Format, das Ihr System erkennt.
(%)	Prozentwert-Platzhalter. Der Ausdruck wird mit 100 multipliziert. Das Prozentzeichen (%) wird an der Stelle in die Position eingefügt, an der es in der Formatzeichenkette erscheint.
(,)	Tausendertrennzeichen. In einigen Darstellungen wird ein Punkt als Tausendertrennzeichen verwendet. Das Tausendertrennzeichen trennt Tausender von Hundertern innerhalb einer Zahl, die vier oder mehr Stellen zur Linken der Dezimalstellen-Trennung besitzt. Die standardmäßige Verwendung des Tausendertrennzeichens wird definiert, wenn das Format ein Tausendertrennzeichen enthält, welches von Zahlzeichen-Platzhaltern umgeben ist (0 oder #). Zwei nebeneinander stehende Tausendertrennzeichen oder ein Tausendertrennzeichen direkt

zur Linken der Dezimalstellen-Trennung (je nachdem, ob eine Dezimalstelle definiert ist, oder nicht) bedeutet eine "Skalierung der Zahl, indem man sie durch 1000 teilt und rundet, falls nötig." Beispielsweise können Sie die Format-Zeichenkette "##0,," verwenden, um 100 Millionen als 100 darzustellen. Zahlen kleiner als 1 Million werden als 0 angezeigt. Zwei nebeneinander stehende Tausendertrennzeichen in einer anderen Position als direkt zur Linken der Dezimalstellen-Trennung stellen eine Spezifizierung der Nutzung eines Tausendertrennzeichens dar. Das Zeichen, welches tatsächlich als Tausendertrennzeichen in den formatierten Ausgabedaten verwendet wird, hängt ab von dem Zahlen-Format, das Ihr System erkennt.

(:) Zeittrennzeichen. Es kann vorkommen, dass in einigen Darstellungen andere Zeichen verwendet werden, um das Zeittrennzeichen darzustellen. Das Zeittrennzeichen trennt Stunden, Minuten, und Sekunden, wenn die Zeitwerte formatiert werden. Das Zeichen, welches tatsächlich als Zeittrennzeichen in den formatierten Ausgabedaten verwendet wird, hängt von Ihren Systemeinstellungen ab.

(/) Datumstrennzeichen. Es kann vorkommen, dass in einigen Darstellungen andere Zeichen verwendet werden, um das Datumstrennzeichen darzustellen. Das Datumstrennzeichen trennt Tag, Monat und Jahr, wenn die Datumswerte formatiert werden. Das Zeichen, welches tatsächlich als Datumstrennzeichen in den formatierten Ausgabedaten verwendet wird, hängt von Ihren Systemeinstellungen ab.

(E- E+) Wissenschaftliches Format. Wenn der Formatausdruck mindestens einen Zahlzeichen-Platzhalter (0 oder #) zur Rechten von E-, E+, e-, oder e+ enthält, so wird die Zahl im wissenschaftlichen Format angezeigt, und E oder e werden zwischen die Zahl und ihren Exponenten eingefügt. Die Anzahl von Zahlzeichen-Platzhaltern zur Rechten bestimmt die Anzahl von Zahlzeichen im Exponenten. Verwenden Sie E- oder e- um ein Minuszeichen neben negative Exponenten zu setzen. Verwenden Sie E+ oder e+ um ein Minuszeichen neben negative Exponenten zu setzen und ein Pluszeichen neben positive Exponenten.

- + $ ( ) Anzeige eines Buchstabensymbol-Zeichens. Um ein Zeichen anzuzeigen, das nicht aufgelistet ist, stellen Sie ihm einen Backslash (\) voran oder stellen Sie es in Anführungsstriche (" ").

(\) Anzeige des nächsten Zeichens in der Formatzeichenkette. Um ein Zeichen anzuzeigen, welches als Buchstabensymbol-Zeichen eine besondere Bedeutung hat, stellen Sie ihm einen Backslash (\) voran. Der Backslash selber wird nicht angezeigt. Die Verwendung eines Backslash bedeutet dasselbe, als stellten Sie das nächste Zeichen in Anführungsstriche (" ").

Um einen Backslash anzuzeigen, verwenden Sie zwei Backslashes (\\). Beispiele von Zeichen, die nicht als Buchstabensymbol-Zeichen angezeigt werden können sind diejenigen, die zur Formatierung von Datum und Uhrzeit verwendet werden (a, c, d, h, m, n, p, q, s, t, w, y, / und :), die Zeichen zur numerischen Formatierung (#, 0, %, E, e, Komma und Punkt), und die Zeichen zur Formatierung von Zeichenketten (@, &, <, >, und !).

("ABC") Zeigen Sie die Zeichenkette innerhalb der Anführungsstriche (" ") an. Um eine Zeichenkette im Format von innerhalb des Codes einzubeziehen, müssen Sie Chr$(34) verwenden, um den Text einzuschließen (34 ist der Zeichencode für ein Anführungszeichen (")).

Datum / Zeit Formatspezifikationssymbole

Zeichen	Beschreibung
(:)	Zeittrennzeichen. Es kann vorkommen, dass in einigen Darstellungen andere Zeichen verwendet werden, um das Zeittrennzeichen darzustellen. Das Zeittrennzeichen trennt Stunden, Minuten, und Sekunden, wenn die Zeitwerte formatiert werden. Das Zeichen, welches tatsächlich als Zeittrennzeichen in den formatierten Ausgabedaten verwendet wird, hängt von Ihren Systemeinstellungen ab.
(/)	Datumstrennzeichen. Es kann vorkommen, dass in einigen Darstellungen andere Zeichen verwendet werden, um das Datumstrennzeichen darzustellen. Das Datumstrennzeichen trennt Tag, Monat und Jahr, wenn die Datumswerte formatiert

werden. Das Zeichen, welches tatsächlich als Datumstrennzeichen in den formatierten Ausgabedaten verwendet wird, hängt von Ihren Systemeinstellungen ab.

c	Zeigt das Datum als ddddd an und die Zeit als ttttt, in dieser Reihenfolge. Zeigt nur dann Datumsinformationen an, wenn die Datums-Seriennummer keine Nachkommastellen hat; Zeitinformationen werden nur angezeigt, wenn kein Integer Anteil vorhanden ist.
d	Zeigt den Tag als Zahl ohne vorangehenden Null an (1-31).
dd	Zeigt den Tag als Zahl mit vorangehender Null an (01-31).
ddd	Zeigt das Datum als Abkürzung an [Sun-Sat (dt. = So-Sa).]
dddd	Zeigt das Datum in ganzen Worten an [Sunday–Saturday (dt. = Sonntag-Samstag)]
ddddd	Zeigt das Datum als komplettes Datum an (inclusive Tag, Monat und Jahr), das in Abhängigkeit von den „Kurzes Datum“-Formateinstellungen Ihres Systems formatiert wird. Für Microsoft Windows lautet das Vorgabeformat für das Kurzdatum m/d/yy.
dddddd	Zeigt eine Datums-Seriennummer als komplettes Datum an (inclusive Tag, Monat und Jahr) das in Abhängigkeit von den „Long Datum“-Formateinstellungen Ihres Systems formatiert wird. Für Microsoft Windows lautet das Vorgabeformat für das Langdatum mmmm dd, yyyy.
w	Zeigt den Wochentag als Zahl an (1 für Sonntag bis 7 für Samstag).
ww	Zeigt die Woche im Jahr als Zahl an (1-54).
m	Zeigt den Monat als Zahl ohne vorangehende Null an (1-12). Wenn m direkt auf h oder hh folgt, wird eher die Minute, als der Monat angezeigt.
mm	Zeigt den Monat als Zahl mit vorangehender Null an (01-12). Wenn m direkt auf h oder hh folgt, wird eher die Minute, als der Monat angezeigt.
mmm	Zeigt den Monat als Abkürzung an (Jan–Dez).
mmmm	Zeigt den Monat mit vollem Monatsnamen an (Januar-Dezember)
q	Zeigt das Jahresquartal als Zahl an (1-4).
y	Zeigt den Tag im Jahr als Zahl an (1-366).
yy	Zeigt das Jahr als zweistellige Zahl an (00–99).
yyyy	Zeigt das Jahr als vierstellige Zahl an (100–9999).
h	Zeigt die Stunde als Zahl ohne vorangehende Null an (0-23).
hh	Zeigt die Stunde als Zahl mit vorangehenden Nullen an (0-23).
n	Zeigt die Minute als Zahl ohne vorangehende Nullen an (0-59).
nn	Zeigt die Minute als Zahl mit vorangehenden Nullen an (0-59).
s	Zeigt die Sekunde als Zahl ohne vorangehende Nullen an (0-59).
ss	Zeigt die Sekunde als Zahl mit vorangehenden Nullen an (0-59).
t t t t t	Zeigt die Zeit als vollständige Zeit an (inklusive Stunde, Minute und Sekunde), formatiert unter Verwendung des Zeittrennzeichens, definiert durch das Zeitformat, das von Ihrem System erkannt wird. Eine vorangehende Null wird angezeigt, wenn die entsprechende Option angewählt ist und die Zeit vor 10:00Uhr morgens oder abends ist. Für Microsoft Windows lautet das Vorgabe-Zeitformat h:mm:ss.
AM/PM	Verwendet die 12-Stunden-Uhr und zeigt für jede Stunde vor dem Mittag ein großbuchstabiges AM an und für jede Stunde zwischen Mittag und 11:59 Uhr am Abend ein großbuchstabiges PM.
am/pm	Verwendet die 12-Stunden-Uhr und zeigt für jede Stunde vor dem Mittag ein kleinbuchstabiges AM an und für jede Stunde zwischen Mittag und 11:59 Uhr am Abend ein kleinbuchstabiges P.M.
A/P	Verwendet die 12-Stunden-Uhr und zeigt für jede Stunde vor dem Mittag ein großbuchstabiges A an und für jede Stunde zwischen Mittag und 11:59 Uhr am Abend ein großbuchstabiges P.
a/p	Verwendet die 12-Stunden-Uhr und zeigt für jede Stunde vor dem Mittag ein kleinbuchstabiges A an und für jede Stunde zwischen Mittag und 11:59 Uhr am Abend ein kleinbuchstabiges P.
AMPM	Verwendet die 12-Stunden-Uhr und zeigt das AM Zeichenketten-Buchstabensymbol bei jeder Stunde vor 12 Uhr Mittag, so wie es in Ihrem System definiert ist. Das PM Zeichenketten-Buchstabensymbol wird für jede Stunde zwischen Mittag und 11:59 am

Abend angezeigt, so wie es in Ihrem System definiert ist. AMPM kann groß oder klein geschrieben werden, aber die Schreibung der angezeigten Zeichenkette entspricht der, die in Ihren Systemeinstellungen definiert ist. Für Microsoft Windows lautet das Vorgabeformat AM/PM.

Verwandte Befehle

Left$, Right$, Str$

FmtStr Beispiel

```
Function SaveData

    String d$, f$, t$

    ' Erstellt den Dateinamen im Format
    ' Monat, Tag, Stunde, Minute
    d$ = Date$
    t$ = Time$
    d$ = d$ + " " + t$
    FmtStr d$, "mmddhhmm", f$
    f$ = f$ + ".dat"
    WOpen f$ as #30
    Print #30, "data"
    Close #30
Fend
```

FolderExists Funktion

F

Überprüft die Existenz eines Ordners.

Syntax

FolderExists(*pathName*)

Parameter

pathName Ein Zeichenkettenausdruck, der den Pfad des zu überprüfenden Ordners enthält. Das Laufwerk kann ebenfalls enthalten sein.

Rückgabewerte

WAHR, wenn die Datei existiert, FALSCH, wenn sie nicht existiert.

Verwandte Befehle

FileExists

Beispiel einer FolderExists Funktion

```
If Not FolderExists("d:\data") Then
    MkDir "d:\data"
EndIf
```

For...Next

S

Die Befehle For...Next werden zusammen verwendet, um eine Schleife herzustellen, in welcher Befehle, die sich zwischen den Befehlen For und Next befinden, mehrfach ausgeführt werden, wie vom Anwender angegeben.

Syntax

For *var* **=** *initValue* **To** *finalValue* **[Step** *increment* **]**
statements
Next [*var*]

Parameter

var	Die Zählvariable, die mit der For...Next Schleife verwendet wird. Diese Variable wird normalerweise als Integer definiert, kann jedoch auch als Realvariable definiert werden.
initValue	Der Anfangswert für den Zähler *var*.
finalValue	Der Endwert des Zählers *var*. Sobald dieser Wert erreicht ist, ist die For...Next Schleife vollständig und die Ausführung wird mit der Anweisung, welche auf den Next-Befehl folgt, fortgesetzt.
increment	Ein optionaler Parameter, der das Zähl-Inkrement für jedes Mal definiert, welches die Next-Anweisung innerhalb der For...Next Schleife ausgeführt wird. Diese Variable kann positiv oder negativ sein. Wenn der Wert jedoch negativ ist, muss der Erstwert der Variable größer sein, als ihr Endwert. Wird der Inkrement-Wert weggelassen, inkrementiert das System automatisch um 1.
statements	Jede gültige SPEL Anweisung kann in die For...Next Schleife eingefügt werden.

Beschreibung

For...Next führt einen Satz von Anweisungen innerhalb einer Schleife eine definierte Anzahl von Malen aus. Die **For** Anweisung stellt den Anfang der Schleife dar. Die **Next** Anweisung ist das Ende der Schleife. Die Male, welche die Anweisungen innerhalb der Schleife ausgeführt werden, werden mithilfe einer Variable gezählt.

Der erste numerische Ausdruck (*initValue)* ist der Erstwert des Zählers. Dieser Wert kann positiv oder negativ sein, solange die Variable *finalValue* und die Step-Inkrementierung einander korrekt entsprechen.

Der zweite numerische Ausdruck (*finalValue)* ist der Endwert des Zählers. Dies ist der Wert der, sobald er erreicht ist, die Beendigung der For...Next Schleife auslöst, und die Steuerung des Programms wird an den nächsten auf den Next-Befehl folgenden Befehl weitergegeben.

Programmanweisungen, die der **For** Anweisung folgen, werden ausgeführt bis ein Next-Befehl erreicht wird. Die Zählervariable (*var*) wird dann durch den Step-Wert inkrementiert, der durch den Parameter *increment* definiert ist. Wird die Step Option nicht genutzt, wird der Zähler um 1 (eins) inkrementiert.

Die Zählervariable (*var*) wird dann mit dem Endwert verglichen. Wenn der Zählerstand kleiner oder gleich dem Endwert ist, werden die Anweisungen, die dem **For** Befehl folgen, erneut ausgeführt. Wenn die Zählervariable größer als der Endwert ist, wird die Ausführung außerhalb der For...Next Schleife verzweigt und fährt mit dem Befehl fort, welcher direkt auf den **Next** Befehl folgt.

Eine Verschachtelung von For...Next Anweisungen wird bis zu einer Tiefe von 16 Leveln unterstützt. Dies bedeutet, dass eine For...Next Schleife in eine andere For...Next Schleife verschachtelt werden kann usw., bis 16 For...Next Schleifen existieren.

HINWEISE

NEGATIVE Step-Werte:

Wenn der Wert der Step-Inkrementierung (*increment*) negativ ist, wird die Zählervariable (*var*) bei jedem Durchlauf durch die Schleife dekrementiert (verringert) und der Erstwert muss größer sein, als der Endwert, damit die Schleife funktioniert.

Eine Variable die auf Next folgt, wird nicht benötigt:

Der Variablenname, der auf den Next-Befehl folgt, kann weggelassen werden. Für Programme, die verschachtelte **For...Next** Schleifen beinhalten wird jedoch empfohlen, den Variablennamen, der auf den Next-Befehl folgt mit einzubeziehen, damit die Schleifen schnell identifiziert werden können.

Verwandte Befehle

Do...Loop, While...Wend

For...Next Beispiel

```
Function fornext
    Integer ctr
    For ctr = 1 to 10
        Go Pctr
    Next ctr

    For ctr = 10 to 1 Step -1
        Go Pctr
    Next ctr
Fend
```

Force_Calibrate Anweisung

S

Setzt Null-Offsets für alle Achsen des aktuellen Kraftsensors.

Syntax

Force_Calibrate

Parameter

On | **Off** Die Drehmomentsteuerung kann ein- oder ausgeschaltet sein.

Beschreibung

Sie sollten **Force_Calibrate** für jeden Sensor aufrufen, wenn ihre Anwendung startet. Dies definiert das Gewicht der Komponenten, die auf dem Sensor montiert sind.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Force Sensing” installiert ist.

Verwandte Befehle

Force_Sensor Anweisung

Beispiel einer Force_Calibrate Anweisung

```
Force_Calibrate
```

Force_GetForce Funktion

F

Gibt die Kraft für eine angegebenen Achse aus.

Syntax

Force_GetForce (*axis)*

Parameter

axis Integer Ausdruzck, welcher die Achse repräsentiert.

Achse	Konstante	Wert
X Kraft	KRAFT_XKRAFT	1
Y Kraft	KRAFT_YKRAFT	2
Z Kraft	KRAFT_ZKRAFT	3
X Drehmoment	KRAFT_XDREHMOMENT	4
Y Drehmoment	KRAFT_YDREHMOMENT	5
Z Drehmoment	KRAFT_ZDREHMOMENT	6

Rückgabewerte

Gibt einen Real-Wert aus.

Beschreibung

Verwenden Sie Force_GetForce um die aktuellen Krafteinstellungen für eine Achse zu lesen. Die Einheiten werden durch den Kraftsensortypen bestimmt.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Force Sensing” installiert ist.

Verwandte Befehle

Force_GetForces

Beispiel einer Force_GetForce Funktion

```
Print Force_GetForce(1)
```

Force_GetForces Anweisung

S

Gibt Kräfte und Drehmomente für alle Kraftsensorachsen in einer Matrix aus.

Syntax

Force_GetForces *array()*

Parameter

array() Real-Matrix mit einer Obergrenze von 6.

Rückgabewerte

Die Matrixelemente werden eingesetzt wie folgt:

Fx Kraft: 1

Fy Kraft: 2

Fz Kraft: 3

Tx Drehmoment: 4

Ty Drehmoment: 5

Tz Drehmoment: 6

Beschreibung

Die **Force_**GetForces Anweisung wird verwendet, um alle Kraft- Und Drehmomentwerte gleichzeitig zu lesen.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Force Sensing” installiert ist.

Verwandte Befehle

Force_GetForce Anweisung

Beispiel einer Force_GetForces Anweisung

```
Real fValues(6)
Force_GetForces fValues()
```

Force_Sensor Funktion

F

Gibt den aktuellen Kraftsensor für den aktuellen Task aus.

Syntax

Force_Sensor

Beschreibung

Force_Sensor gibt die aktuelle Sensornummer für den aktuellen Task aus. Wenn ein Task startet, wird die Sensornummer automatisch auf 1 gesetzt.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Force Sensing” installiert ist.

Verwandte Befehle

Force_Sensor Anweisung

Beispiel einer Force_Sensor Funktion

```
var = Force_Sensor
```

Force_Sensor Anweisung

F

Stellt den aktuellen Kraftsensor für den aktuellen Task ein.

Syntax

Force_Sensor *sensorNumber*

Parameter

sensorNumber Integer-Ausdruck, welcher die Sensornummer repräsentiert.

Beschreibung

Wenn mehrere Kraftsensoren im selben System verwendet werden, müssen Sie den aktuellen Kraftsensor einstellen, bevor Sie andere Kraftmessungsbefehle verwenden.

Wenn Ihr System nur über einen Sensor verfügt, müssen Sie die Force_Sensor Anweisung nicht verwenden, da die Vorgabe-Sensornummer 1 ist.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Force Sensing” installiert ist.

Verwandte Befehle

Force_Sensor Funktion

Beispiel einer Force_Sensor Funktion

```
Force_Sensor 1
```

Force_SetTrigger Anweisung

F

Stellt die Kraftauslösung für den Till-Befehl ein.

Syntax

Force_SetTrigger *axis, Threshold, CompareType*

Parameter

axis Integer-Ausdruck, der die gewünschte Kraftsensorachse enthält.

Achse	Konstante	Wert
X Kraft	KRAFT_XKRAFT	1
Y Kraft	KRAFT_YKRAFT	2
Z Kraft	KRAFT_ZKRAFT	3
X Drehmoment	KRAFT_XDREHMOMENT	4
Y Drehmoment	KRAFT_YDREHMOMENT	5
Z Drehmoment	KRAFT_ZDREHMOMENT	6

Threshold Real-Ausdruck, der den gewünschten Schwellwert in Einheiten für den verwendeten Sensor enthält.

CompareType 0 = kleiner als, 1 = größer als

Beschreibung

Um eine Bewegung mithilfe eines Kraftsensors zu stoppen, müssen Sie den Trigger einstellen und dann Till Force in ihrer Bewegungsanweisung verwenden.

Sie können den Trigger für mehrere Achsen setzen. Rufen Sie Force_SetTrigger für jede Achse auf. Setzen Sie den Schwellenwert auf 0, um eine Achse zu deaktivieren.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die Option “Force Sensing” installiert ist.

Verwandte Befehle

Force_Calibrate

Beispiel einer Force_SetTrigger Anweisung

```
Setzt einen Trigger, wenn die Kraft der Z-Achse niedriger als -1 ist, um
die Bewegung zu stoppen.
Force_SetTrigger 3, -1, 0
SpeedS 3
AccelS 5000
Move Place Till Force
```

Force_TC Anweisung

S

Schaltet die Drehmomentsteuerung Ein oder Aus und zeigt den aktuellen Modusstatus an.

Syntax

Force_TC [{ **On** | **Off** }]

Parameter

On | **Off** Die Drehmomentsteuerung kann Ein oder Aus geschaltet sein.

Rückgabewerte

Zeigt die aktuelle **Force_TC** Moduseinstellung an, wenn der Parameter weggelassen wird.

Beschreibung

Verwenden Sie Force_TC, um die Drehmomentsteuerung ein- oder auszuschalten. Ebenso müssen über Force_TCLim Drehmomentbegrenzungen eingestellt werden, bevor die Drehmomentsteuerung wirksam wird.

Wenn Sie sich unter Drehmomentsteuerung zu einem Punkt bewegen und dann Force_TC Off ausführen, kann es sein, dass ein Fehler ausgegeben wird, mit dem Hinweis, dass die aktuelle Position nicht aufrecht erhalten werden konnte. Das liegt daran, dass sich der Roboter ohne Drehmomentsteuerung nicht zur richtigen Position bewegen kann.

Verwandte Befehle

Force_TCLim Anweisung

Beispiel einer Force_TC Anweisung

```
Speed 5
' Setzt die Drehmomentbegrenzung der Z-Achse auf 20 Prozent
Force_TCLim -1, -1, 20, -1
Force_TCSpeed 5
Force_TC On
Go Place
Force_TC Off
```

Force_TCLim Funktion

F

Gibt die aktuelle Drehmomentbegrenzungs-Einstellung für einen definierten Punkt aus.

Syntax

Force_TCLim(*jointNumber*)

Parameter

jointNumber Integer-Ausdruck, welcher die Achsnummer repräsentiert, für die die Drehmomentbegrenzungs-Einstellung ausgegeben werden soll.

Rückgabewerte

Ganze Zahl, welche die aktuelle Drehmomentbegrenzung enthält. -1 zeigt, dass die Drehmomentbegrenzung deaktiviert ist..

Verwandte Befehle

Force_TCLim Anweisung

Beispiel einer Force_TCLim Funktion

```
Print "The current torque limit for Z is: ", Force_TCLim(3)
```

Force_TCLim Anweisung

S

Stellt die Drehmomentbegrenzung für alle Achsen für die Drehmomentsteuerung ein.

Syntax

Force_TCLim [*j1TorqueLimit*, *j2TorqueLimit*, *j3TorqueLimit*, *j4TorqueLimit*]

Parameter

j1TorqueLimit	Integer-Ausdruck, der den Drehmomentbegrenzungswert für Achse 1 in Prozentangaben repräsentiert. Setzen Sie den Wert auf -1 um die Anweisung zu deaktivieren.
j2TorqueLimit	Integer-Ausdruck, der den Drehmomentbegrenzungswert für Achse 2 in Prozentangaben repräsentiert. Setzen Sie den Wert auf -1 um die Anweisung zu deaktivieren.
j3TorqueLimit	Integer-Ausdruck, der den Drehmomentbegrenzungswert für Achse 3 in Prozentangaben repräsentiert. Setzen Sie den Wert auf -1 um die Anweisung zu deaktivieren.
j4TorqueLimit	Integer-Ausdruck, der den Drehmomentbegrenzungswert für Achse 4 in Prozentangaben repräsentiert. Setzen Sie den Wert auf -1 um die Anweisung zu deaktivieren.

Rückgabewerte

Zeigt die Drehmomentbegrenzungswerte, die für jede Achse definiert sind, wenn die Parameter weggelassen werden.

Beschreibung

Die Drehmomentbegrenzungs-Einstellungen sind wirksam, wenn der TC Modus eingeschaltet ist. Diese Einstellungen stellen einen Prozentwert des gesamten Drehmoments dar.

Wenn die Einstellung zu niedrig ist, kann es sein, dass der Roboter sich nicht bewegen kann. In diesem Fall wird der Bewegungsbefehl stoppen, bevor das Ziel erreicht ist.

Nachdem Force_TCLim eingestellt wurde, muss Force_TC On erneut ausgeführt werden. Die Begrenzungen werden automatisch deaktiviert (auf -1 gesetzt) wenn Reset, Motor On, oder SLock auftreten.

Verwandte Befehle

Force_TC Anweisung, Force_TCLim Funktion

Beispiel einer Force_TCLim Anweisung

```
Speed 5
' Setzt die Drehmomentbegrenzung der Z-Achse auf 20 Prozent
Force_TCLim -1, -1, 20, -1
Force_TCSpeed 5
Force_TC On
Go Place
```

Force_TCSpeed Funktion

F

Gibt die aktuelle Geschwindigkeitseinstellung für die Drehmomentsteuerung aus.

Syntax
Force_TCSpeed
Rückgabewerte
Aktuelle Force_TCSpeed-Einstellung.

Verwandte Befehle
Force_TCSpeed Anweisung

Beispiel einer Force_TCSpeed Funktion

```
Integer var
var = Force_TCSpeed
```

Force_TCSpeed Anweisung

F

Stellt die Geschwindigkeit für die Drehmomentsteuerung ein.

Syntax

Force_TCSpeed [*speed*]

Parameter

speed Integer-Ausdruck, der die Geschwindigkeit für die Drehmomentsteuerung repräsentiert. Der Bereich liegt zwischen 1 und 100.

Beschreibung

Force_TCSpeed ist die Geschwindigkeit, in der der Roboter sich nach Erreichen der Drehmomentbegrenzung bewegt.

Verwandte Befehle

Force_TCLim, Force_TCSpeed Funktion

Beispiel einer Force_TCSpeed Anweisung

```
Speed 5
' Setzt die Drehmomentbegrenzung der Z-Achse auf 20 Prozent
Force_TCLim -1, -1, 20, -1
Force_TCSpeed 5
Force_TC On
Go Place
```

Function...Fend

S

Eine Funktion ist eine Gruppe von Programmanweisungen, welche eine **Funktion**-Anweisung als erste und eine Fend-Anweisung als letzte Anweisung beinhaltet.

Syntax

Function *funcName* [(*argList*)] [**As** *type*]
statements
Fend

Parameter

funcName	Name, der einer spezifischen Gruppe von Anweisungen gegeben wird, die zwischen die Befehle **Function** und **Fend** eingebunden werden. Der Funktionsname muss alphanumerische Zeichen enthalten und darf bis zu 32 Zeichen lang sein. Unterstriche sind ebenfalls gestattet.
argList	Optional. Liste von Variablen, die Argumente repräsentieren, die an die Funktionsprozedur weitergegeben werden, wenn diese aufgerufen wird. Multiple Variablen werden durch Kommata getrennt.

Das arglist-Argument hat die folgende Syntax:

[{**ByRef | ByVal**}] *varName* [( )] **As** *type*

ByRef	Optional. Spezifizieren Sie ByRef, wenn Sie möchten, dass Veränderungen im Variablenwert von der Calling-Funktion gesehen werden.
ByVal	Optional. Spezifizieren Sie ByVal, wenn Sie *nicht* möchten, dass Veränderungen im Variablenwert von der Calling-Funktion gesehen werden. Dies ist die Vorgabeeinstellung.
varName	Erforderlich. Name der Variable, welche das Argument repräsentiert; folgt den Standardkonventionen für Variablenbenennung.
As *type*	Erforderlich. Der Argumenttyp muss definiert werden.

Rückgabewerte

Wert, dessen Datentyp mit dem **As**-Ausdruck am Ende der Funktionsdeklaration definiert wird.

Beschreibung

Die Anweisung **Funktion** zeigt den Beginn einer Gruppe von SPEL Anweisungen an. Um anzuzeigen, wo eine Funktion endet, wird die Anweisung **Fend** verwendet. Alle Anweisungen, die sich zwischen **Function** und **Fend** befinden, werden als Teile der Funktion betrachtet.

Die durch **Function...Fend** definierte Kombination von Anweisungen kann als ‚Behältnis' betrachtet werden, in dem alle Anweisungen, die sich zwischen den Anweisungen **Function** und **Fend** befinden zu der Funktion gehören. In einer Programmdatei können multiple Funktionen vorkommen.

Verwandte Befehle

Call, Fend, Halt, Quit, Return, Xqt

Function...Fend Beispiel

Das folgende Beispiel zeigt 3 Funktionen innerhalb einer einzigen Datei. Die Funktionen mit dem namen *task2* und *task3* werden als Hintergrundtasks (Background Tasks) ausgeführt, während der Haupttask mit Namen *Main* im Vordergrund ausgeführt wird.

```
Function main
  Xqt 2, task2 'Führt task2 im Hintergrund aus
  Xqt 3, task3 'Führt task3 im Hintergrund aus
  '... weitere Anweisungen
Fend

Function task2
  While TRUE
    On 1
    On 2
    Off 1
    Off 2
  Wend
Fend

Function task3
  While TRUE
    On 10
    Wait 1
    Off 10
  Wend
Fend
```

GetCurrentUser$ Funktion

F

Gibt den aktuellen EPSON RC+ Anwender aus.

Syntax

GetCurrentUser$

Rückgabewerte

Zeichenkette, die die aktuelle Anwender-logID enthält.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die “Security” Option („Sicherheitsoption“) installiert ist.

Verwandte Befehle

LogIn Anweisung

Beispiel einer GetCurrentUser$ Funktion

```
String currUser$

currUser$ = GetCurrentUser$
```

Global Anweisung

S

Deklariert Variablen mit globalem Gültigkeitsbereich. Globale Variablen sind frei zugänglich.

Syntax

Global [**Preserve**] *dataType varName* [**(***subscripts***)**] [**,** *varName* [**(***subscripts***)**] **,** ...]

Parameter

Preserve	Optional. Wenn Preserve definiert ist, werden die Werte der Variable beibehalten, indem ihr Inhalt auf Diskette gespeichert wird. Die Werte werden nur durch Erase gelöscht, oder wenn die Anzahl der Dimensionen geändert wird. Die Werte werden über Sessions oder Projektänderungen hinaus beibehalten.
dataType	Datentypen wie z.B. Boolean, Integer, Long, Real, Double, Byte, oder String.
varName	Variablenname. Namen dürfen bis zu 24 Zeichen lang sein.
subscripts	Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt **(***dim1* [, *dim2* [, *dim3*]]**)** *dim1*, *dim2*, *dim3* können ein Integer zwischen 0-32767 sein.

Beschreibung

Globalvariablen sind Variablen, die in mehr als einer Datei desselben Projektes verwendet werden können. Sie werden gelöscht, wann immer eine Funktion vom Run-Fenster oder Operator-Fenster aus gestartet wird, es sei denn, sie sind mit der Preserve-Option deklariert.

Globale Variablen können im Online-Fenster verwendet werden, sobald die Projektgenerierung abgeschlossen ist.

Globale Variablen können ebenfalls mit der VB-Guide Option verwendet werden.

Es wird empfohlen, den Namen globaler Variablen ein "g_"-Präfix voranzustellen, um Globale im Programm leichter erkennbar zu machen. Zum Beispiel:

```
Global Long g_PartsCount
```

Verwandte Befehle

Byte, Double, Integer, Long, Real, String

Beispiel einer Global Anweisung

Das folgende Beispiel zeigt zwei verschiedene Programmdateien. Die erste Programmdatei definiert einige Globalvariablen und initialisiert sie. Die zweite Datei verwendet eben diese Globalvariablen.

DATEI1 (MAIN.PRG)

```
Global Integer status1
Global Real numsts

Function Main
    Integer I

    status1 = 10
```

Das folgende Beispiel zeigt zwei verschiedene Programmdateien. Die erste Programmdatei definiert einige Globalvariablen und initialisiert sie. Die zweite Datei verwendet auch diese Globalvariablen.

DATEI1 (MAIN.PRG)

```
Global Integer g_Status
Global Real g_MaxValue

Function Main

    g_Status = 10
    g_MaxValue = 1.1
    .
    .
Fend
```

DATEI2 (TEST.PRG)

```
Function Test

    Print "status1 = , g_Status
    Print "MaxValue = , g_MaxValue
    .
    .
Fend
```

Go Anweisung

> S

Bewegt den Arm in einer PTP-Bewegung von seiner aktuellen Position zum definierten Punkt oder zur Position X,Y,Z,U. Der Go-Befehl kann eine beliebige Kombination von einer bis zu vier Achsen gleichzeitig bewegen.

Syntax

Go destination [**CP**] [searchExpr] [!...!]

Parameter

destination	Der Zielort einer Bewegung, die einen Punktausdruck verwendet.
CP	Optional. Spezifiziert die Continuous Path (CP / kontinuierlicher Weg) -Bewegung.
searchExpr	Optional. Ein Till oder Find Ausdruck. **Till** | **Find** **Till Sw**(*expr*) = {**Ein** | **Aus**} **Find Sw**(*expr*) = {**Ein** | **Aus**}
!...!	Optional. Parallelbearbeitungsanweisungen können hinzugefügt werden, um E/A und andere Befehle während der Bewegung auszuführen.

Beschreibung

Go bewegt die Achsen eins bis vier des Roboterarmes gleichzeitig in einer PTP-Bewegung. Das Ziel des Go-Befehls kann in vielerlei Hinsicht definiert werden:

- Verwenden eines spezifischen Punktes, auf welchen man sich zubewegt. Zum Beispiel: **Go P1**.
- Verwenden einer expliziten Koordinatenposition, auf welche man sich zubewegt. Zum Beispiel: **Go 50, 400, 0, 0.**
- Verwenden eines Punktes mit Koordinatenversatz. Zum Beispiel: **Go P1 +X(50)**.
- Verwenden eines Punktes mit einem andere Koordinatenwert. Zum Beispiel: **Go P1 :X(50)**.

Der Speed Befehl bestimmt die Armgeschwindigkeit für die Bewegung, welche durch den Go-Befehl initiiert wurde. Der Accel-Befehl definiert die Beschleunigung.

Der **Go**-Befehl veranlasst den Arm immer dazu, vor dem Erreichen der Zielposition mit dem Abbremsen zu beginnen.

Hinweise

Der Unterschied zwischen Go und Move

Sowohl der Move-, als auch der **Go**-Befehl veranlassen den Roboterarm zu einer Bewegung. Der primäre Unterschied zwischen den beiden Befehlen ist jedoch, dass der **Go**-Befehl eine PTP-Bewegung verursacht, während der Move-Befehl den Arm geradlinig bewegt. Der **Go**-Befehl wird verwendet, wenn es dem Anwender primär um die Ausrichtung des Arms bei Erreichen des Punktes geht. Der **Move**-Befehl wird verwendet, wenn es wichtig ist, den Pfad des Roboterarms zu steuern, während er sich bewegt.

Der Unterschied zwischen Go und Jump

Sowohl der Jump-Befehl, als auch der **Go**-Befehl veranlassen den Roboterarm zu einer PTP-Bewegung. Der Jump-Befehl hat jedoch ein zusätzliches Feature. Jump veranlasst den Endeffektor des Roboters dazu, zunächst bis zum LimZ-Wert zu steigen, sich dann in horizontaler Richtung zu bewegen, bis er sich über dem Zielpunkt befindet und schließlich abwärts zum Zielpunkt. Dies gestattet es, den Jump-Befehl zu verwenden, um das Umfahren von Hinderniss zu garantieren und, was wichtiger ist, um die Zykluszeiten für Bestückungsbewegungen zu verbessern.

Korrekte Geschwindigkeits- und Beschleunigungs-Befehle mit Go

Die Befehle Speed und Accel werden verwendet, um Geschwindigkeit und Beschleunigung des Manipulators während der **Go**-Bewegung zu definieren. Achten Sie darauf, dass die Befehle Speed und Accel sich auf PTP-Bewegungen beziehen (wie der für den **Go** Befehl), während die Befehle SpeedS und AccelS für linearinterpolierte und kreisinterpolierte Bewegungen verwendet werden.

Verwenden von Go mit der optionalen Till Bedingung

Die optionale Till-Bedingung gestattet es dem Anwender eine Bedingung zu definieren, damit der Roboter an einer Zwischenposition bis zum Halt abbremst, noch bevor er die durch den **Go**-Befehl verursachte Bewegung beendet. Wenn die Till-Bedingung nicht erfüllt wird, bewegt sich der Roboter zu seiner Zielposition. Der Go-Befehl mit der Till-Bedingung kann auf zwei Arten und Weisen benutzt werden, wie im folgenden beschrieben:

(1) Go-Befehl mit Till-Bedingung
Überprüft, ob die aktuelle Till-Bedingung erfüllt wird. Wenn sie erfüllt wird, wird dieser Befehl durch Verzögerung und Anhalten des Roboters an einer Zwischenposition beendet.

(2) Go-Befehl mit Till-Bedingung, Sw(Eingangsnummer)-Bedingung, und Eingangsbedingung
Diese Version des Go-Befehls mit Till-Bedingung gestattet es dem Anwender, die Till-Bedingung in derselben Zeile wie den Go-Befehl zu spezifizieren, anstatt die aktuelle Definition zu verwenden, die im Vorangegangenen für Till definiert wurde. Die spezifizierte Bedingung ist ganz einfach eine Gegenprüfung zu einem der Eingänge. Dies wird durch Verwenden des Sw-Befehls erreicht. Der Anwender kann überprüfen, ob die Eingänge aus- oder eingeschaltet sind und den Arm dazu veranlassen, anzuhalten, je nachdem, welche Bedingung spezifiziert ist. Dieses Feature arbeitet ähnlich wie eine Unterbrechung (Interrupt), bei der eine Bewegung unterbrochen (gestoppt) wird, sobald die Eingangs-Bedingung erfüllt ist. Wenn die Bedingung während der Roboterbewegung nie erfüllt wird, erreicht der Arm erfolgreich den durch *destination* definierten Punkt.

Verwenden von Go mit der optionalen Find Bedingung

Die optionale Find Bedingung gestattet es dem Anwender, eine Bedingung zu definieren, die den Roboter veranlasst, während der Bewegung, die durch den **Go**-Befehl veranlasst wurde, eine Position aufzuzeichnen. Der Go-Befehl mit der Find-Bedingung kann auf zwei Arten und Weisen benutzt werden, wie im folgenden beschrieben:

(1) Go-Befehl mit Find-Bedingung:
Überprüft, ob die aktuelle Find-Bedingung erfüllt wird. Wird sie erfüllt, so wird die aktuelle Position im Sonderpunkt FindPos gespeichert.

(2) Go-Befehl mit Find-Bedingung, Sw(Eingangsnummer)-Bedingung, und Eingangsbedingung:
Diese Version des Go-Befehls mit Find-Bedingung gestattet es dem Anwender, die Find-Bedingung in derselben Zeile wie den Go-Befehl zu spezifizieren, anstatt die aktuelle Definition zu verwenden, die im Vorangegangenen für Find definiert wurde. Die spezifizierte Bedingung ist ganz einfach eine Gegeprüfung zu einem der Eingänge. Dies wird durch Verwenden des Sw-Befehls erreicht. Der Anwender kann überprüfen, ob die Eingänge aus- oder eingeschaltet sind und veranlassen, dass die aktuelle Position unter dem Sonderpunkt **FindPos** gespeichert wird.

Der Go-Befehl verursacht immer eine Verzögerung bis zum Stillstand

Der **Go**-Befehl veranlasst den Arm immer dazu, vor dem Erreichen der Zielposition mit dem Abbremsen zu beginnen.

Potentielle Fehler

Der Versuch, sich außerhalb des Roboter-Arbeitsbereiches zu bewegen
Wenn mit dem **Go**-Befehl explizite Koordinaten verwendet werden, müssen Sie sicherstellen, dass die definierten Koordinaten sich innerhalb des Roboter-Arbeitsbereiches befinden. Jeglicher Versuch, den Roboter außerhalb dieses gültigen Arbeitsbereiches zu bewegen, ruft eine Fehlermeldung hervor.

Verwandte Befehle

!...! Parallelbearbeitung, Accel, Find, Jump, Move, Pass, Pn= (Point Assignment / Punktzuordnung), Pulse, Speed, Sw, Till

Go Beispiel

Das folgende Beispiel zeigt eine einfache PTP-Bewegung zwischen den Punkten P0 und P1 und die darauf folgende geradlinige Rückkehr zum Punkt P0. Später im Programm bewegt sich der Arm geradlinig zu Punkt P2, bis Eingang #2 einschaltet. Wenn Eingang #2 während der Bewegung einschaltet, dann verzögert sich der Arm bis zum vollständigen Stopp, bevor er Punkt P2 erreicht und der nächste Programmbefehl ausgeführt wird.

Funktionsbeispiel

```
  Integer i

  Home
  Go P0
  Go P1
  For i = 1 to 10
    Go P(i)
  Next i
  Go P2 Till Sw(2) = On
  If Sw(2) = On Then
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
  Else
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
  End If
Fend
```

Einige Syntax-Beispiele vom Online-Fenster sehen aus wie folgt:

```
>Go P* +X(50)       ' Bewegt den Arm nur in die X-Richtung, 50 mm aus
der
                    ' aktuellen Position
>Go P1              ' Einfaches Beispiel, um den Arm nach Punkt P1 zu
bewegen
>Go P1 :U(30)       ' Bewegt den Arm nach Punkt 1, verwendet aber +30
als Position,
                    ' zu welcher sich die U-Achse bewegen soll
>Go P1 /L           ' Bewegt den Arm zu P1, stellt aber sicher, dass der
Arm
                    ' in der linken Position anhält.
>Go 50, 450, 0, 30  ' Bewegt den Arm in die Position X=50, Y=450, Z=0,
U=30

<Ein weiteres Code-Beispiel>
Till Sw(1)=0 And Sw(2) = On  ' Legt die Till-Bedingungen für die
                             ' Eingängen 1 & 2 fest
Go P1 Till                   ' Stoppt den Arm, wenn die aktuelle Till-,
                             ' die in der vorangegangenen Zeile
definiert wurde, erfüllt ist.
Go P2 Till Sw(2) = ON        ' Stoppt, wenn der Eingang 2 eingeschaltet
ist
Go P3 Till                   ' Stoppt den Arm, wenn die aktuelle Till-
Bedingung,
                             ' die in der vorangegangenen Zeile
definiert wurde, erfüllt ist.
```

GoSub...Return

S

GoSub überträgt die Programmsteuerung an ein Unterprogramm. Sobald das Unterprogramm vollständig ist, kehrt die Programmsteuerung wieder zu der Zeile zurück, die auf den **GoSub** Befehl folgt, welcher das Unterprogramm initiiert hatte.

Syntax

GoSub {*lineNum* | *label*}

{*lineNum* | *label***:**}
statements
Return

Parameter

lineNum	Dies ist die Zeilennummer, zu der die Programmausführung wechselt, wenn der **GoSub** Befehl ausgeführt wird. Gültige Zeilennummern liegen zwischen 0001 und 32767. Jegliche gültige Zeilennummer ist zulässig.
label	Anstatt eine Zeilennummer zu nutzen kann der Anwender auch ein Label definieren (empfohlene Lösung). Wenn der Anwender ein Label definiert (an Stelle der Zeilennummer) wird die Programmausführung zu der Zeile wechseln, in der sich dieses Label befindet. Das Label kann bis zu 24 Zeichen lang sein. Das erste Zeichen muss jedoch ein Buchstabe sein (keine Zahl).

Beschreibung

Der **GoSub** Befehl bewirkt, dass die Programmsteuerung an eine durch den Anwender definierte Anweisungs-Zeilennummer oder zu einem gleichermaßen definierten Label verzweigt wird. Das Programm führt dann die Anweisung in dieser Zeile aus und fährt mit der Ausführung in den darauf folgenden Zeilennummern fort, bis es auf einen Return-Befehl trifft. Der Return-Befehl veranlasst die Programmsteuerung, zurück zu der Zeile zu wechseln, welche direkt auf diejenige folgt, die **GoSub** ursprünglich initiiert hatte. (D.h., der **GoSub** Befehl veranlasst die Ausführung eines Unterprogramms, wonach die Ausführung zu der Anweisung zurückkehr, die auf den **GoSub** Befehl folgt.) Stellen Sie sicher, dass jedes Unterprogramm mit Return beendet wird. Diese Vorgehensweise veranlasst die Programmausführung, zu der Zeile zurückzukehren, die auf den **GoSub** Befehl folgt.

Potentielle Fehler

Verzweigen zu einer nicht existierenden Anweisung

Wenn der **GoSub** Befehl versucht, die Steuerung an eine nicht existente Zeilennummer oder Label zu verzweigen, wird Fehler Nr. 8 ausgegeben.

Der Verschachtelungslevel für GoSub-Befehle ist zu tief

GoSub Befehle dürfen bis zu 16 Level tief verschachtelt werden. Das bedeutet, dass ein weiterer **GoSub** Befehl von innerhalb eines Unterprogramms ausgeführt werden kann (usw.) – bis zu 16 Mal. Sobald jedoch 16 verschachtelte **GoSub** Befehle ausgeführt sind, wird der Versuch, einen 17. **GoSub** Befehl auszuführen, in der Ausgabe des Fehlers Nr. 7 resultieren.

Return-Befehl ohne GoSub-Befehl gefunden

Ein Return-Befehl wird verwendet, um aus einem Unterprogramm ins Ursprungsprogramm "zurückzukehren", das den **GoSub** Befehl ausgegeben hat. Wenn ein Return-Befehl gefunden wird, ohne dass dem ein **GoSub** Befehl voran gegangen ist, wird Fehler Nr. 3 ausgegeben. Ein alleinstehender Return-Befehl ist bedeutungslos, da das System nicht weiß, wohin es zurückkehren soll.

Verwandte Befehle

GoTo, OnErr, Return

Beispiel einer GoSub Anweisung

Das folgende Beispiel zeigt eine einfache Funktion, die einen GoSub-Befehl verwendet, um zu einem Label zu verzweigen, einige E/A-Befehle auszuführen und dann zurückzukehren.

```
Function main
    Integer var1, var2

    GoSub checkio  'GoSub unter Verwendung von Label
    On 1
    On 2
    Exit Function

checkio:    'Hier startet die Subroutine
    var1 = In(0)
    var2 = In(1)
    If var1 = 1 And var2 = 1 Then
        On 1
    Else
        Off 1
    EndIf
    Return  'Hier endet die Subroutine
Fend
```

Goto Anweisung

S

Der **GoTo** Befehl bewirkt, dass die Programmsteuerung ohne Bedingung an eine designierte Anweisungs-Zeilennummer oder zu einem Label verzweigt wird.

Syntax

GoTo { *linenum* | *label* }

Parameter

lineNum	Dies ist die Zeilennummer, zu der die Programmausführung wechselt, wenn der **GoTo** Befehl ausgeführt wird. Gültige Zeilennummern liegen zwischen 0001 und 32767. Jegliche gültige Zeilennummer ist zulässig.
label	Anstatt eine Zeilennummer zu nutzen, kann der Anwender auch ein Label definieren. Wenn der Anwender im "go to" Programm ein Label spezifiziert, springt die Programmausführung zu der Zeile, in der sich das Label befindet. Das Label kann bis zu 32 Zeichen lang sein. Das erste Zeichen muss jedoch ein Buchstabe sein (keine Zahl).

Beschreibung

Der **GoTo** Befehl bewirkt, dass die Programmsteuerung an eine durch den Anwender definierte Anweisungs-Zeilennummer oder zu einem gleichermaßen definierten Label verzweigt wird. Das Programm führt dann die Anweisung in der betreffenden Zeile aus und fährt dann mit der Ausführung von dort aus fort. **GoTo** wird am häufigsten für den Sprung zu einem Exit Label auf Grund einer Fehlermeldung verwendet.

Einschränkung

Verwendung zu vieler GoTo-Anweisungen

Bitte seien Sie mit der **GoTo**-Anweisung vorsichtig, da die Verwendung zu vieler **GoTos** in einem Programm letzteres schwer verständlich machen können. Die generelle Regel lautet, so wenig **GoTos** zu verwenden, wie möglich. Einige **GoTo**s sind fast immer notwendig. Es kann jedoch leicht zu Problemen führen, wenn man sich mit zu vielen **GoTo** Anweisungen innerhalb des Quellcodes bewegt.

Potentielle Fehler

Verzweigen zu einer nicht existierenden Anweisung

Wenn der **GoTo** Befehl versucht, die Steuerung an eine nicht existente Zeilennummer oder Label zu verzweigen, wird Fehler Nr. 8 ausgegeben.

Verwandte Befehle

GoSub, OnErr

Beispiel einer GoTo Anweisung

Das folgende Beispiel zeigt eine einfache Funktion, die einen GoTo-Befehl verwendet, um zu einem Zeilenlabel zu verzweigen.

```
Function main

    If Sw(1) = Off Then
        GoTo mainAbort
    EndIf
    Print "Input 1 was On, continuing cycle"
    .
    .
    Exit Function

mainAbort:
    Print "Input 1 was OFF, cycle aborted!"
Fend
```

Halt Anweisung

S

Unterbricht vorübergehend die Ausführung eines spezifizierten Tasks.

Syntax

Halt *taskIdentifier*

Parameter

taskIdentifier	Taskname oder Integer-Ausdruck, der die Tasknummer repräsentiert. Ein Taskname ist der Funktionsname, der in einer Xqt-Anweisung verwendet wird, oder eine Funktion, die vom Run- oder vom Operator-Fenster aus gestartet wird. Wenn ein Integer-Ausdruck verwendet wird, liegt der Bereich zwischen 1 und 32.

Beschreibung

Halt unterbricht vorübergehend die Ausführung des aktuellen Tasks, wie durch den Tasknamen oder die -Nummer definiert.

Verwenden Sie Resume, um mit der Ausführung des Tasks an der Stelle fortzufahren, wo er unterbrochen wurde. Verwenden Sie Quit, um die Ausführung eines Tasks vollständig abzubrechen. Um den Taskstatus anzuzeigen, klicken Sie auf das *Task Manager* Icon auf der EPSON RC+ Werkzeugleiste, um den Taskmanager auszuführen.

Verwandte Befehle

Quit, Resume, Xqt

Beispiel einer Halt Anweisung

Das folgende Beispiel zeigt eine Funktion mit Namen "flicker", die von *Xqt* gestartet wird, temporär von *Halt* gestoppt und danach von Resume wieder aufgenommen wird.

```
Function main
    Xqt 2, flicker   'Führt die Flicker-Funktion aus

  Do
    Wait 3           'Führt den Task Flicker 3 Sekunden lang aus
    Halt flicker

    Wait 3           'Hält den Task Flicker 3 Sekunden lang an
    Resume flicker

  Loop
Fend
    '

Function flicker
  Do
    On 1
    Wait 0.2
    Off 1
    Wait 0.2
  Loop
Fend
```

Here Anweisung

S

Teacht einen Roboterpunkt an der aktuellen Position ein.

Syntax
Here *point*
Parameter
point Roboterpunkt.

Verwandte Befehle
Here Funktion

Beispiel einer Here Anweisung

```
Here P1    ' Ist gleichbedeutend wie P1=P*
Here pick    ' Ist gleichbedeutend wie P*
```

Here Funktion

F

Gibt die aktuelle Roboterposition als Punkt aus.

Syntax

Here

Rückgabewerte

Ein Punkt, der die aktuelle Roboterposition repräsentiert.

Beschreibung

Here wird verwendet, um die aktuelle Position des aktuellen Manipulators auszugeben. Dies ist äquivalent zu P*.

Verwandte Befehle

Here Anweisung

Beispiel einer Here Funktion

```
P1 = Here
```

Hex$ Funktion

> F

Gibt eine Zeichenkette aus, welche die angegebene Zahl im Hexadezimalformat darstellt.

Syntax

Hex$*(number)*

Parameter

number Integer-Ausdruck.

Rückgabewerte

Gibt eine Zeichenkette aus, welche die ASCII-Darstellung der Zahl im Hexadezimalformat enthält.

Beschreibung

Hex$** gibt eine Zeichenkette aus, welche die angegebene Zahl im Hexadezimalformat darstellt. Jedes Zeichen besteht aus 0-9 oder A-F. **Hex$ wird insbesondere verwendet, um die Ergebnisse der Stat Funktion über einen RS232 Port zu prüfen.

Verwandte Befehle

Str$, Stat, Val

Beispiel einer Hex$ Funktion

```
> print hex$(stat(0))
A00000
> print hex$(255)
FF
```

Hofs Anweisung

> S

Zeigt die Offset-Pulse zwischen Encoderursprung und Home-Sensor an oder stellt sie ein.

Syntax

Hofs [*j1Pulses, j2Pulses, j3Pulses, j4Pulses*]

Parameter

j1Pulses	Integer-Ausdruck, der die Offset-Pulse für Achse 1 darstellt.
j2Pulses	Integer-Ausdruck, der die Offset-Pulse für Achse 2 darstellt.
j3Pulses	Integer-Ausdruck, der die Offset-Pulse für Achse 3 darstellt.
j4Pulses	Integer-Ausdruck, der die Offset-Pulse für Achse 4 darstellt.

Rückgabewerte

Zeigt die aktuellen Hofs-Werte ohne Verwendung von Parametern an.

Beschreibung

Hofs zeigt die Home-Position Offset-Pulse an oder stellt sie ein. Wenn der Roboter die Achsen zu ihren Home-Positionen bringt, überprüft das Steuergerät die Position mittels Home-Sensor und Encoder Z-Phase. Normalerweise sind die Werte des Home-Sensors und der Encoder Z-Phase jedoch unterschiedlich. **Hofs** wird daher verwendet, um diesen Offset-Wert zu definieren. (**Hofs** definiert den Offset vom Encoder-Nullpunkt zum mechanischen Nullpunkt.)

Obwohl die Bewegungssteuerung des Roboters auf dem Nullpunkt des Encoders basiert, der auf jeden Achsmotor montiert ist, stimmen Encoder-Nullpunkt und mechanischer Nullpunkt des Roboters nicht notwendigerweise überein. Der Korrektur-Pulse des **Hofs** Offset-Pulse wird verwendet, um eine Softwarekorrektur am mechanischen Nullpunkt, basierend auf dem Encoder-Nullpunkt, durchzuführen.

Hinweis

Hofs-Werte SOLLTEN NICHT geändert werden, es sei denn, dies ist unumgänglich notwendig

Die Hofs-Werte werden vor der Auslieferung korrekt definiert. Es besteht die Gefahr, dass die unnötige Veränderung des Hofs-Wertes in Positionsfehlern und unvorhersehbaren Bewegungen resultiert. **Wir legen Ihnen daher dringend nahe**, die Hofs-Werte nicht zu ändern, es sei denn, es ist absolut notwendig.

Automatische Berechnung von Hofs-Werten

Um **Hofs**-Werte automatisch zu berechnen bewegen Sie den Arm in die gewünschte Kalibrierungsposition und führen Sie den Calib**-**Befehl aus. Die Steuerung berechnet die Hofs-Werte dann automatisch auf Basis der CalPls-Pulse-Werte und der Pulse-Werte der Kalibrierungsposition.

Speichern und Wiederherstellen von Hofs

Hofs kann mit den Befehlen Mkver und SetVer aus dem Wartungsdialog im Tools-Menü gespeichert und wiederhergestellt werden.

Verwandte Befehle

Calib, CalPls, Home, Hordr, HTest, Mcal, Ver

Beispiel einer Hofs Anweisung

Dies sind einfach Beispiele im Online-Fenster, bei denen im ersten Fall der Home-Offset für die 1. Achse auf -545 eingestellt wird, für die 2. Achse der Home-Offset Wert auf 514, und die Home-Offset-Werte für Z- und U-Achse beide auf 0. Dann werden die aktuellen Home-Offset Werte angezeigt.

```
> hofs  -545, 514, 0, 0

> hofs
-545, 514, 0, 0
>
```

Hofs Funktion

F

Gibt die Offset-Pulse aus, die für die softwaremäßige Nullpunktkorrektur verwendet werden.

Syntax

Hofs(*jointNumber*)

Parameter

jointNumber Integer-Ausdruck, der die Achsnummer darstellt, für die der Hofs-Wert ausgegeben werden soll.

Rückgabewerte

Der Offset-Pulse-Wert (Integer-Wert, in Pulsen).

Verwandte Befehle

Calib, CalPls, Home, Hordr, HTest, Mcal, Ver

Beispiel einer Hofs Funktion

Dieses Beispiel benutzt die **Hofs**-Funktion in einem Programm:

```
Function DisplayHofs
   Integer i

   Print "Hofs settings:"
   For i = 1 To 4
      Print "Joint ", i, " = ", Hofs(i)
   Next i
Fend
```

Home Anweisung

> S

Bewegt den Roboterarm in die durch den Anwender definierte Home-Position.

Syntax

Home

Beschreibung

Führt bei niedriger Geschwindigkeit eine PTP-Bewegung zur Home- (Standby-) Position durch, die durch den HomeSet-Befehl definiert ist, in der Homing-Reihenfolge, die durch *Hordr* definiert ist. Es ist sehr wichtig zu beachten, dass der Home-Befehl den Roboter nicht kalibriert.

Die Kalibrierung wird über den Mcal-Befehl für Roboter mit inkrementellen Encodern ausgeführt. Der **Home**-Befehl bewegt den Arm nur zu einer durch HomeSet definierten Standby-Position bei niedriger Geschwindigkeit.

Üblicherweise kehrt die Z-Achse als erstes zur HomeSet Position zurück, dann kehren die Achsen 1, 2 und U gleichzeitig zu ihrer entsprechenden HomeSet-Koordinatenposition zurück. Der Hordr-Befehl kann die Reihenfolge der Rückkehr der Achsen zu ihren Home-Positionen verändern.

Hinweis

Ausgang des Home Status:

Wenn sich der Roboter in seiner Home-Position befindet, ist der Home-Ausgang der Steuerung eingeschaltet.

Potentielle Fehler

Homingversuch ohne definierte HomeSet-Werte

Der Versuch, den **Home**-Befehl am Roboter auszuführen, ohne die HomeSet-Werte zu setzen, resultiert in Ausgabe des Fehlers Nr. 143.

Verwandte Befehle

Hofs, HTest, HOrdr

Beispiel einer Home-Anweisung

Der Home-Befehl kann in einem Programm wie diesem verwendet werden:

```
Function InitRobot
   Reset
   If Motor = Off Then
      Motor On
   EndIf
   Home
Fend

Or it can be issued from the Monitor window like this:

> home
>
```

Homeset Anweisung

> S

Definiert die Home-Position und zeigt sie an.

Syntax

HomeSet [*j1Pulses*, *j2Pulses*, *j3Pulses*, *j4Pulses*]

Parameter

j1Pulses Der Home-Position Encoder-Pulse-Wert für Achse 1.

j2Pulses Der Home-Position Encoder-Pulse-Wert für Achse 2.

j3Pulses Der Home-Position Encoder-Pulse-Wert für Achse 3.

j4Pulses Der Home-Position Encoder-Pulse-Wert für Achse 4.

Rückgabewerte

Zeigt die Pulse-Werte an, die für die aktuelle Home-Position definiert sind, wenn die Parameter weggelassen werden.

Beschreibung

Gestattet es dem Anwender, eine neue Home- (Standby-) Position zu definieren, indem er die Pulse-Werte für jede der vier Achsen eingibt.

Hinweise

Der Home-Befehl kalibriert den Roboter nicht:

Dieser Hinweis bezieht sich ausschließlich auf Roboter mit Inkrementalencoder.

Obgleich der **HomeSet**-Befehl die Home-Position Encoder-Werte einstellt, ist es sehr wichtig zu beachten, dass der Home-Befehl den Roboter nicht kalibriert. Zur Kalibrierung des Roboters wird der Mcal-Befehl verwendet. (Wenn die Versorgungsspannung zum ersten Mal eingeschaltet wird, muss der Roboter über den Mcal-Befehl kalibriert werden.)

Verinit und sein Effekt auf die HomeSet-Werte:

Die Ausführung des Verinit-Befehls löscht die aktuellen **HomeSet**-Werte.

Potentielle Fehler

Homingversuch ohne definierte HomeSet-Werte:

Der Versuch, den **HomeSet**-Befehl am Roboter auszuführen, ohne die HomeSet-Werte zu setzen, resultiert in Ausgabe des Fehlers Nr. 143.

Der Versuch, HomeSet-Werte anzuzeigen, ohne dass HomeSet-Werte definiert sind:

Der Versuch Home-Position Pulse-Werte anzuzeigen, ohne dass HomeSet-Werte definiert sind, resultiert in Ausgabe des Fehlers Nr. 143.

Verwandte Befehle

Home, Hordr, Mcal, Pls

HomeSet Beispiel

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt:

```
> home
!!Error 143 : Home position is not defined

> homeset
!!Error 143 : Home position is not defined

> homeset 0,0,0,0  'Setzt die Home-Position auf 0,0,0,0
> homeset
  0  0
  0  0

> home  'Der Roboter kehrt zur Home-Position 0,0,0,0 zurück
```

Die Pls-Funktion wird verwendet, um die aktuelle Position des Armes als Home-Position zu definieren.

```
> homeset Pls(1), Pls(2), Pls(3), Pls(4)
```

Homeset Funktion

F

Gibt die Pulse-Werte der Home-Position für die angegebene Achse aus.

Syntax

HomeSet(*jointNumber)*

Parameter

jointNumber Integer-Ausdruck, der die Achsnummer darstellt, für die der HomeSet-Wert ausgegeben werden soll.

Rückgabewerte

Gibt den Pulse-Wert der Home-Position der Achse aus. Wenn die *jointNumber* 0 beträgt, wird 1 ausgegeben, wenn HomeSet ausgeführt wurde und 0, wenn HomeSet nicht ausgeführt wurde.

Verwandte Befehle

Homeset Anweisung

Beispiel einer HomeSet Funktion

Dieses Beispiel verwendet die **Homeset**-Funktion in einem Programm:

```
Function DisplayHomeset

   Integer i

   If HomeSet(0) = FALSE Then
      Print "HomeSet is not defined"
   Else
      Print "HomeSet values:"
      For i = 1 To 4
         Print "J", i, " = ", HomeSet(i)
      Next i
   EndIf
Fend
```

Hordr Anweisung

> S

Definiert die Reihenfolge, in welcher die Achsen in ihre Home-Position zurückkehren oder zeigt sie an.

Syntax

Hordr [*step1, step2, step3, step4*]

Parameter

step1 Bitmuster, aus dem hervorgeht, welche Achsen während des 1. Schrittes des Homing-Prozesses in ihre Home-Position zurückgeführt werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 1. Schrittes in ihre Home-Position zurückgeführt werden. (bzgl der Definition von Bitmustern: siehe unten)

step2 Bitmuster, aus dem hervorgeht, welche Achsen während des 2. Schrittes des Homing-Prozesses in ihre Home-Position zurückgeführt werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 2. Schrittes in ihre Home-Position zurückgeführt werden. (bzgl der Definition von Bitmustern: siehe unten)

step3 Bitmuster, aus dem hervorgeht, welche Achsen während des 3. Schrittes des Homing-Prozesses in ihre Home-Position zurückgeführt werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 3. Schrittes in ihre Home-Position zurückgeführt werden. (bzgl der Definition von Bitmustern: siehe unten)

step4 Bitmuster, aus dem hervorgeht, welche Achsen während des 4. Schrittes des Homing-Prozesses in ihre Home-Position zurückgeführt werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 4. Schrittes in ihre Home-Position zurückgeführt werden. (bzgl der Definition von Bitmustern: siehe unten)

Rückgabewerte

Zeigt die aktuellen *Home Order* Einstellungen an, wenn die Parameter weggelassen werden.

Beschreibung

Hordr definiert die Bewegungsreihenfolge für den Home-Befehl. (D.g. die Anweisung definiert, welche Achse zuerst in seine Home-Position zurückgeführt werden soll, welches als 2., 3., usw.)

Der Zweck des **Hordr-**Befehls ist es, dem Anwender zu gestatten, die Reihenfolge zu ändern, in der die Achsen in ihre Home-Position zurückgeführt werden sollen. Die Homing-Reihenfolge ist in vier unterschiedliche Schritte unterteilt. Der Anwender verwendet **Hordr** dann, um die spezifischen Achsen zu definieren, die sich während eines jeden Schrittes in die Home-Position bewegen. Es ist wichtig, sich zu verdeutlichen, dass mehr als eine Achse definiert werden kann, welche sich während eines einzigen Schrittes in die Home-Position bewegen soll. Dies bedeutet, dass alle vier Achsen potentiell zur selben Zeit in ihre Home-Position zurückgeführt werden können. Es wird jedoch empfohlen, im Normalfall zunächst die Z-Achse zu ihrer Home-Position zu bewegen (in Step/Schritt 1) und die anderen Achsen in den darauf folgenden Steps/Schritten folgen zu lassen. (Siehe unten stehenden Hinweis)

Der **Hordr** Befehl setzt voraus, dass ein Bitmuster für jeden der vier Schritte definiert wurde. Da es vier Achsen gibt, wird jeder Achse ein spezifisches Bit zugeordnet. Wenn das Bit gesetzt ist (1) (für einen spezifischen Schritt), wird die entsprechende Achse sich in ihre Home-Position bewegen. Wenn das Bit nicht gesetzt ist (0), wird sich die entsprechende Achse während dieses Schrittes nicht in ihre Home-Position bewegen. Die Achsen-Bitmuster werden zugeordnet wie folgt:

Achse:	**1. Achse (T1)**	**2. Achse (T2)**	**3. Achse (Z)**	**4. Achse (U)**
Bitnummer:	Bit 0	Bit 1	Bit 2	Bit 3
Binärcode:	&B0001	&B0010	&B0100	&B1000

Hinweise

Der Unterschied zwischen MCordr und Hordr

obgleich die Befehle MCordr und **Hordr** auf den ersten Blick recht ähnlich aussehen, gibt es zwischen ihnen einen Hauptunterschied, den man verstehen muss. MCordr wird verwendet, um die

Achsenreihenfolge für die Roboterkalibrierung zu definieren (wird zusammen mit Mcal benutzt) während **Hordr** verwendet wird, um die Achsenreihenfolge für das Homing zu definieren (wird zusammen mit dem Home Befehl verwendet).

Vorgegebene Homing-Reihenfolge (werksseitige Einstellung)

Werksseitig ist die Homing-Reihenfolge dahingehend eingestellt, dass die Z-Achse im 1. Schritt in sein Home-Position zurückgeführt wird. Im 2. Schritt werden dann die Achsen 1, 2, und U gleichzeitig in ihre Home-Position zurückgeführt. (Die Schritte 3 und 4 werden in der Vorgabekonfiguration nicht verwendet). Die vorgegebenen **Hordr**-Werte lauten wie folgt:

```
Hordr &B0100, &B1011, 0, 0
```

Der Verinit Befehl (Zurücksetzen der vorgegebenen Hordr-Werte)

Die **Hordr-**Werte werden durch Ausführen des Verinit-Befehls auf die o.a. Werte zurückgesetzt.

Hordr Werte werden beibehalten

Die **Hordr**-Werte werden dauerhaft gespeichert und nicht geändert, bis sie entweder vom Anwender geändert werden, oder ein Verinit-Befehl ausgeführt wird.

Die Z-Achse sollte im Normalfall zuerst in ihre Home-Position zurückgeführt werden.

Der Grund, warum die Z-Achse als Erste (und ausschließlich) bewegt werden soll liegt darin, dass das Roboterwerkzeug über die Arbeitsfläche angehoben werden soll, bevor mit einer Horizontalbewegung begonnen wird. Dies trägt dazu bei, zu verhindern, mit dem Roboterwerkzeug während des Homing-Prozesses im Arbeitsbereich gegen etwas zu stoßen.

Verwandte Befehle

Mcal, MCordr, Home, HomeSet

Beispiel einer HOrdr Anweisung

Im Folgenden sehen Sie einige Online-Fenster-Beispiele:

Dieses Beispiel instruiert den Roboterarm, die Homing-Reihenfolge wie folgt zu definieren: Z-Achse im ersten Schritt, Achse 1 im zweiten Schritt, Achse 2 im dritten Schritt, und dann die U-Achse im vierten Schritt. Die Reihenfolge ist in Bitmustern spezifiziert.

```
>hordr  &B0100, &B0001, &B0010, &B1000
```

Dieses Beispiel instruiert den Roboterarm die Z-Achse als Erste in ihre Home-Position zurückzuführen (in Step/Schritt 1), und danach die Achsen 1, 2 und U gleichzeitig (&B1011, oder 11 als Dezimalwert) in Step/Schritt 2. Die Homing-Reihenfolge ist mit Dezimalwerten spezifiziert.

```
>hordr  4, 11, 0, 0
```

Dieses Beispiel zeigt die aktuelle Homing-Reihenfolge in Hexadezimalzahlen.

```
>hordr
4, 11, 0, 0
>
```

HOrdr Funktion

> F

Gibt den Hordr-Wert für einen spezifizierten Schritt aus.

Syntax
Hordr(*stepNumber*)
Parameter
stepNumber Integer-Ausdruck, welcher den auszugebenden Hordr-Schritt repräsentiert.
Rückgabewerte
Integer, der den Hordr-Wert für den angegebenen Schritt enthält.
Verwandte Befehle
Mcal, MCordr, Home, HomeSet
Beispiel einer HOrdr Funktion

```
Integer a
a = Hordr(1)
```

Hour Anweisung

>

Zeigt die Betriebszeit der Steuerung an.

Syntax

Hour

Beschreibung

Zeigt die Zeit an, die der PC bereits eingeschaltet ist und SPEL ausführt. (Kumulierte Betriebszeit) Die Zeit wird immer in ganzen Stunden angezeigt.

Verwandte Befehle

Time

Hour Beispiel

Das folgende Beispiel wird vom Online-Fenster aus ausgeführt:

```
> hour
2560
```

Hour Funktion

F

Gibt die Betriebszeit der Steuerung aus.

Syntax

Hour

Rückgabewerte

Gibt die kumulierte Betriebszeit des Steuergerätes aus (Real-Zahl, in Stunden).

Verwandte Befehle

Time

Beispiel einer Hour Funktion

```
Print "Number of controller operating hours: ", Hour
```

HTest Anweisung

>

Zeigt die Anzahl Pulse vom Einschalten des Homesensors bis zur Z-Phase des Encoders.

Syntax

HTest

Rückgabewerte

Der Pulse-Zählstand für jede Achse wird angezeigt.

Beschreibung

Bei Ausführung des Mcal-Befehls bewegt sicht der Roboterarm von der Position, in der der Kalibrierungssensor ausgeschaltet ist (sollte der Kalibrierungssensor eingeschaltet sein, bewegt sich der Arm, um den Sensor abzuschalten) zu der Position, in der der Sensor eingeschaltet wird, und fährt mit seiner Bewegung fort, bis er die Encoder-Z-Phase innerhalb einer Rotation findet, dann wird der Arm angehalten. Der **HTest-**Befehl zeigt den Pulse-Zählstand für jede Achse an, die diese Bewegung durchläuft.

Vor jedem Roboterversand wird der Pulse-Zählstand, der einige Fehler in der Home-Sensor Detektion beinhaltet, zu Ihrer Sicherheit angepasst

Potentielle Fehler

Der HTest-Wert überschreitet seinen Bereich:

Wenn sich die **HTest-**Werte außerhalb des festgesetzten, erlaubten Bereichs bewegen, tritt ein Fehler auf.

Verwandte Befehle

Hofs, Mcal

Beispiel einer HTest Anweisung

Zeigt die aktuellen HTest-Werte im Online-Fenster an:

```
>htest
344, 386, 254, 211
>
```

HTest Funktion

F

Gibt den HTEST-Wert für das angegebene Glenk aus.

Syntax

HTest(*jointNumber*)

Parameter

jointNumber Ein Integer-Ausdruck, der die Achsnummer repräsentiert, für welche der HTest-Wert ausgegeben werden soll.

Rückgabewerte

Gibt den HTEST-Wert der spezifizierten Achse (Integer-Wert, in Pulsen) aus.

Verwandte Befehle

HTest Anweisung

Beispiel einer HTest Funktion

Dieses Beispiel benutzt die **HTest**-Funktion in einem Programm:

```
Function DisplayHTest

   Integer i

   Print "HTest values:"
   For i = 1 To 4
      Print HTest(i)
   Next i
Fend
```

If…Then…Else…EndIf Anweisung S

Führt Befehle auf der Basis einer angegebenen Bedingung aus.

Syntax

(1) **If** *condition* **Then**
stmtT1
.
.
[ElseIf *condition* **Then]**
stmtT1
.
.
[Else]
stmtF1
.
.
EndIf

(2) **If** *condition* **Then** *stmtT1* **[;** *stmtT2***...]** **[Else** *stmtF1* **[;** *stmtF2***...]]**

Parameter

condition	Jegliche gültige Bedingung, die ein WAHRes (beliebige Zahl außer 0) oder FALSCHes Resultat ausgibt (als 0 ausgegeben). (S.u. für Beispielbedingungen)
stmtT1	Wird ausgeführt, wenn die Bedingung WAHR ist. (Mehrfachanweisungen können in geblocktem **If...Then...Else** -Stil hierher geschrieben werden.)
stmtF1	Wird ausgeführt, wenn die Bedingung FALSCH ist. (Mehrfachanweisungen können in geblocktem **If...Then...Else** -Stil hierher geschrieben werden.)

Beschreibung

(1) **If...Then...Else** führt stmtT1 aus, usw., wenn die bedingte Anweisung WAHR ist. Ist die Bedingung FALSCH, wird stmtF1, usw., ausgeführt. Der **Else-**Teil des **If...Then...Else** Befehls ist optional. Wird die **Else**-Anweisung weggelassen und die bedingte Anweisung ist FALSCH, wird die Anweisung ausgeführt, welche auf die EndIf-Anweisung folgt. Für geblockte **If...Then...Else** Anweisungen muss die EndIf Anweisung den Block schließen, egal ob ein **Else** genutzt wird, oder nicht.

Eine Verschachtelung von bis zu 20 Leveln wird für die geblockte **If...Then...Else**-Anweisung unterstützt.

(2) **If...Then...Else** kann auch nicht-geblockt verwendet werden. Dies gestattet es, Anweisungen für **If...Then...Else** in dieselbe Zeile zu schreiben. Bitte bechten Sie, dass die EndIf–Anweisung nicht benötigt wird, wenn **If...Then...Else** in nicht geblockter Form verwendet wird. Wenn die in dieser Zeile spezifizierte Bedingung erfüllt wird (WAHR), werden die Anweisungen zwischen Then und Else ausgeführt. Wenn die Bedingung nicht erfüllt wird(FALSCH), werden die Anweisungen ausgeführt, die auf **Else** folgen. Der **Else-**Teil von **If...Then...Else** wird nicht benötigt. Existiert kein **Else**-Schlüsselwort, wird die Steuerung an die nöchste Anweisung im Programm weitergegeben, wenn die **If**-Bedingung FALSCH ist.

Der logische **Wert** der bedingten Anweisung ist jede Zahl außer 1 (Eins) wenn sie WAHR ist, und 0, wenn sie FALSCH ist.

Hinweise

Beispielbedingungen:

a = b	:a ist gleich b
a < b	:b ist größer als a
a >= b	:a ist größer/gleich b
a <> b	:a ist ungleich b
a > b	:b ist kleiner als a
a <= b	:a ist kleiner/gleich b

Logische Operationen wie And, Or und Xor können ebenfalls verwendet werden.

Verwandte Befehle

Else, Select...Case, While...Wend

Beispiel einer If/Then/Else Anweisung

<Einzeilige If...Then...Else>

Das folgende Beispiel zeigt eine einfach Funktion, die einen Eingang überprüft, um festzustellen, ob ein bestimmter Ausgang ein- oder ausgeschaltet werden soll. Dieser Task könnte ein Background- (Hintergrund-) I/O Task sein, der fortwährend läuft.

```
Function main
    Do
       If Sw(0) = 1 Then On 1 Else Off 1
    Loop
Fend
```

<**Geblockte If...Then...Else>**

Das folgende Beispiel zeigt eine einfach Funktion, die einige Eingänge überprüft und den Status dieser Eingänge druckt.

```
If Sw(0) = 1 Then Print "Input0 ON" Else Print "Input0 OFF"
'
If Sw(1) = 1 Then
    If Sw(2) = 1 Then
       Print "Input1 On and Input2 ON"
    Else
       Print "Input1 On and Input2 OFF"
    EndIf
Else
    If Sw(2) = 1 Then
       Print "Input1 Off and Input2 ON"
    Else
       Print "Input1 Off and Input2 OFF"
    EndIf
EndIf
```

<Beispiele mit anderer Syntax>

```
If x = 10 And y = 3 Then GoTo 50
If test <= 10 Then Print "Test Failed"
If Sw(0) = 1 Or Sw(1) = 1 Then Print "Everything OK"
```

In Funktion

F

Gibt den Status des spezifizierten Eingangsports aus. Jeder Port enthält 8 Eingangskanäle.

Syntax

In(*portNumber*)

Parameter

portNumber Zahl zwischen 0 und 63, die einen der 64 I/O Ports darstellt. Bitte beachten Sie, dass jeder Port 8 Eingangskanäle enthält, was damit 512 Eingangskanäle ausmacht.

Rückgabewerte

Gibt einen Integer-Wert zwischen 0 und 255 aus. Der Rückgabewert beträgt 8 Bits, wobei jedes Bit einem Eingangskanal entspricht.

Beschreibung

In gibt Ihnen die Möglichkeit, die Werte von 8 Eingangskanälen gleichzeitig zu betrachten. Der In-Befehl kann verwendet werden, um den Status der 8 Eingangskanäle in einer Variable zu speichern. Alternativ kann er auch mit dem Wait-Befehl verwendet werden, um zu warten, bis eine spezifische Bedingung erfüllt ist, die mehr als einen Eingangskanal einschließt.

Da 8 Kanäle zur gleichen Zeit geprüft werden, rangieren die Rückgabewerte zwischen 0 und 255. Bitte beachten Sie die Tabelle unten, um zu sehen, wie die Rückgabewerte den einzelnen Eingabekanälen entsprechen.

Das System unterstützt standardmäßig 16 Eingänge und 16 Ausgänge für jede Drive Unit. Dies bedeutet, dass die Ports 0 und 1 des In-Befehls für jedes System zur Verfügung stehen. Wenn drei Drive Units verwendet werden, stehen die Ports 0 bis 5 zur Verfügung. Die Ports 6 - 63 stehen zur Verfügung, sobald der Anwender zusätzliche I/O–Platinen in die Steuerung installiert, mit 512 zur Verfügung stehenden Eingängen und 512 Ausgängen.

Eingangskanal-Ergebnis (unter Verwendung von Port #0)

Rückgabewerte	7	6	5	4	3	2	1	0
1	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
5	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	On (Ein)
15	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)	On (Ein)	On (Ein)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Eingangskanal-Ergebnis (unter Verwendung von Port #3)

Rückgabewerte	31	30	29	28	27	26	25	24
3	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
7	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)	On (Ein)
32	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Einschränkung

Der Unterschied zwischen In und In $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischen den In und In $ versteht. Der In $ - Befehl arbeitet mit dem ***internen Merker*** und hat keinerlei Auswirkungen auf den Hardware I/O des Roboters. Der In-Befehl arbeitet mit den 512 Hardware Eingabekanälen. Diese Hardwarekanäle sind getrennte Eingänge, die mit systemexternen Geräten kommunizieren.

Der Unterschied zwischen In und Sw

Der Sw-Befehl gestattet es dem Anwender, jeweils einen Hardware-Eingangskanal zu lesen. Der Rückgabewert von Sw ist entweder eine 1 oder eine 0, was zeigt, dass der Hardware-Eingangskanal entweder ein- oder ausgeschaltet ist. Der Sw-Befehl kann jeden der 512 Hardware-Eingänge einzeln prüfen. Der In-Befehl ist dem Sw-Befehl sehr ähnlich, da auch er verwendet wird, um den Status der Hardware-Eingabekanäle zu überprüfen. Es gibt jedoch einen deutlichen Unterschied. Der In-Befehl überprüft gleichzeitig 8 Hardware-Eingabekanäle, wohingegen der Sw-Befehl, wie oben gesehen, immer nur einen einzigen Kanal überprüfen kann. Er gibt einen Wert zwischen 0 und 255 aus, der dem Anwender anzeigt, welcher der 8 I/O Kanäle ein- und welcher ausgeschaltet ist.

Verwandte Befehle

In $, InBCD, Off, Off $, On, On $, OpBCD, Oport, Out, Out $, Sw, Sw $, Wait

Beispiel einer In Funktion

Das unten folgende Beispiel gehen wir davon aus, dass die Eingangskanäle 28,29,30 und 31 alle mit sensorischen Geräten verbunden sind, sodas die Anwendung nicht startet, bis jedes dieser Geräte ein ON-Signal ausgibt und damit Startbereitschaft signalisiert. Das Programmbeispiel erhält den aktuellen Wert der letzten 8 Eingänge und stellt vor dem Fortfahren sicher, dass die Kanäle 28, 29, 30, und 31 eingeschaltet sind. Wenn sie nicht eingeschaltet sind (d.h. sie geben einen Wert von 1 aus) wird eine Fehlermeldung an den Operator ausgegeben und der Task angehalten.
Im Programm wird die Variable "var1" mit der Zahl 239 verglichen, weil der Rückgabewert der eingeschalteteten Eingänge 28, 29,30 und 31 (IN (3)) größer als 240 ist. (Die Eingänge 24, 25, 26, und 27 werden in diesem Fall vernachlässigt, so dass jeglicher Wert zwischen 240 und 255 dem Programm gestattet, fortzufahren.)

```
Function main
    Integer var1
    var1 = In(3)  'Holt die letzten 8 Eingänge
    If var1 > 239 Then
        Go P1
        Go P2
        'Hier werden anderen Bewegungs-Anweisungen ausgeführt
        '.
        '.
    Else
        Print "Error in initialization!"
        Print "Sensory Inputs not ready for cycle start"
        Print "Please check inputs 28,29,30 and 31 for"
        Print "proper state for cycle start and then"
        Print "start program again"
    EndIf
Fend
```

Es ist nicht möglich, Eingänge vom Online-Fenster aus zu setzen, aber man kann sie überprüfen. Für die folgenden Beispiele wird davon ausgegangen, dass die Eingabekanäle 1, 5, 15, und 30 eingeschaltet sind.. Alle anderen Eingänge sind aus.

```
> print In(0)
34
> print In(1)
128
> print In(2)
0
> print In(3)
64
```

In($n) Funktion

F

Gibt den Status des angegebenen Merkerports aus. Jeder Port enthält 8 S/W Merker.

Syntax

In($*portNumber*)

Parameter

portNumber Zahl zwischen 0 und 63, die einen der 64 S/W Merkerports darstellt. Bitte beachten Sie, dass jeder Port 8 S/W Merker enthält, was sich zu einer Gesamtzahl von 512 S/W Merkern addiert.

Hinweis: Das Dollarzeichen "$" muss vor die I/O-Nummer gestellt werden um anzuzeigen, dass dies ein Merker ist.

Rückgabewerte

Gibt einen Integer-Wert zwischen 0 und 255 aus. Der Rückgabewert beträgt 8 Bits, wobei jedes Bit einem S/W Merker entspricht.

Beschreibung

In $ gibt Ihnen die Möglichkeit, die Werte von 8 S/W Merkern gleichzeitig zu betrachten. Der **In $**-Befehl kann verwendet werden, um den Status der 8 S/W-Bits in einer Variable zu speichern. Alternativ kann er auch mit dem Wait-Befehl verwendet werden, um zu warten, bis eine spezifische Bedingung erfüllt ist, die mehr als ein I/O-Bit einschließt.

Da 8 Kanäle zur gleichen Zeit geprüft werden, rangieren die Rückgabewerte zwischen 0 und 255. Bitte beachten Sie die Tabelle unten, um zu sehen, wie die Rückgabewerte den einzelnen S/W Merkern entsprechen.

Merker-Ergebnis (unter Verwendung von Port #0)

Rückgabewerte	7	6	5	4	3	2	1	0
1	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
5	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	On (Ein)
15	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)	On (Ein)	On (Ein)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Merker-Ergebnis (unter Verwendung von Port #31)

Rückgabewerte	255	254	253	252	251	250	249	248
3	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
7	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)	On (Ein)
32	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Einschränkung

Der Unterschied zwischen In und In $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischen den In und In $ versteht. Der **In $** -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf den Hardware I/O des Roboters. Der In-Befehl arbeitet mit den 512 Hardware Eingangskanälen. Diese Hardwarekanäle sind getrennte Eingänge, die mit systemexternen Geräten kommunizieren.

Der Unterschied zwischen $ and Sw $

Der Sw $ -Befehl gestattet es dem Anwender, den Wert eines S/W Merkers zu lesen. Der Rückgabewert von Sw $ ist entweder eine 1 oder eine 0, was zeigt, dass der Merker entweder ein- oder ausgeschaltet ist. Sw $ kann jeden der 512 S/W Merker einzeln prüfen. Der **In $** -Befehl ist dem Sw$-Befehl sehr ähnlich, da auch er verwendet wird, um den Status der S/W Merker zu überprüfen. Es gibt jedoch einen deutlichen Unterschied. Der In$-Befehl überprüft gleichzeitig 8 S/W Merker-Kanäle, wohingegen der Sw$-Befehl, wie oben gesehen, immer nur einen einzigen Kanal überprüfen kann. **In $** gibt einen Wert zwischen 0 und 255 aus, der dem Anwender anzeigt, welcher der 8 Merker ein- und welcher ausgeschaltet ist.

Verwandte Befehle

In, INBCD, Off, Off $, On, On $, OpBCD, Oport, Out, Out $, Sw, Sw $, Wait

In $ Beispiel

Das Programmbeispiel unten liest den Wert der ersten 8 S/W Merker und stellt vor dem Fortfahren sicher, dass momentan alle 8 I/O auf 0 gesetzt sind. Wenn sie nicht auf 0 stehen, wird eine Fehlermeldung an den Operator ausgegeben und der Task angehalten.

```
Function main
    Integer var1

    var1 = In($0)  'Holt die ersten 8 S/W E/A Werte
    If var1 = 0 Then
        Go P1
        Go P2
    Else
        Print "Error in initialization!"
        Print "First 8 S/W I/O channels were not all set to 0"
    EndIf
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> out $0, 1
> print In($0)
1
> on $1
> print In($0)
3
> out $31,3
> print In($31)
3
> off $249
> print In($31)
1
>
```

Einschränkung

Der Unterschied zwischen InBCD und In

Die Befehle **InBCD** und In sind in der SPEL Sprache sehr ähnlich. Es gibt jedoch einen Hauptunterschied zwischen den beiden. Dieser Unterschied wird im Folgenden aufgezeigt.

- Der **InBCD**-Befehl verwendet das BCD-Format, um das Format des Rückgabewertes für die 8 Eingänge zu definieren. Da das BCD-Format den Gebrauch der Werte &HA, &HB, &HC, &HD, &HE oder &HF ausschließt, können nicht alle Kombinationen für die 8 Eingänge verwendet werden.
- Der In-Befehl ist dem InBCD-Befehl sehr ähnlich, außer dass In es gestattet, den Rückgabewert für alle 8 Eingänge zu verwenden. (D.h. 0 bis 255 im Vergleich zu 0-99 für InBCD) Dies gestattet es, dass alle möglichen Kombinationen für die 8 Bit-Eingangsgruppen gelesen werden.

Verwandte Befehle

In, Off, Off $, On, On $, OpBCD, Oport, Out, Out $, Sw, Sw $, Wait

InBCD Beispiel

Einige einfache Beispiele vom Online-Fenster sehen aus wie folgt:

Nehmen wir an, dass an den Eingängen 0, 4, 10, 16, 17, und 18 Spannung anliegt (Die restlichen Eingänge sind ohne Spannung).

```
> Print InBCD(0)
11
> Print InBCD(1)
04
> Print InBCD(2)
07
>
```

Inertia Anweisung

>

Spezifiziert die vollständige Massenträgheit und Exzentrizität für den aktuellen Roboter.

Syntax

Inertia [*loadInertia* [*, eccentricity*]]

Parameter

loadInertia Optional. Real-Ausdruck, der das totale Trägheitsmoment in kgm^2 um das Zentrum von Achse #4 definiert, inklusive Endeffektor und Teil.

eccentricity Optional. Real-Ausdruck, der die Exzentrizität in mm um das Zentrum von Achse #4 definiert, inklusive Endeffektor und Teil.

Rückgabewerte

Wenn die Parameter weggelassen werden, so werden die aktuellen Trägheits-Parameter angezeigt.

Beschreibung

Die **Inertia**-Anweisung wird von Robotern ab der E2-Serie unterstützt. Wenn **Inertia** mit Robotern älteren Typs verwendet wird, treten Fehler auf.

Verwenden Sie die **Inertia-**Anweisung, um das vollständige Trägheitsmoment für die Last an Achse #4 zu definieren. Dies gestattet es dem System, Beschleunigung, Verzögerung und Servoverstärkung für Achse #4 akkurater zu kompensieren. Sie können auch die Entfernung vom Zentrum der Achse #4 zum Schwerpunkt von Endeffektor und Teil definieren, indem Sie den Parameter *eccentricity* verwenden.

Verwandte Befehle

Inertia Funktion

Beispiel einer Inertia Anweisung

```
Inertia 0.02, 1
```

Inertia Funktion

F

Gibt den Wert des Trägheits-Parameters aus.

Syntax

Inertia(*paramNumber***)**

Parameter

paramNumber Integer-Ausdruck, der die folgenden Werte haben kann:

0: Veranlasst die Funktion, 1 auszugeben, wenn der Roboter Trägheitsparameter unterstützt, oder 0, wenn sie nicht unterstützt werden.

1: Veranlasst die Funktion, die Lastträgheit in kgm2 auszugeben.

2: Veranlasst die Funktion, die Exzentrizität in mm auszugeben.

Rückgabewerte

Verwandte Befehle

Inertia Anweisung

Beispiel einer Inertia Funktion

```
Real loadInertia, eccentricity

loadInertia = Inertia(1)
eccentricity = Inertia(2)
```

InPos Funktion

F

Gibt den aktuellen Positionsstatus des spezifizierten Roboters aus.

Syntax

InPos [(*robotNumber*)]

Parameter

robotNumber	Optional. Definiert, für welchen Roboter der Positionsstatus ausgegeben werden soll. Wird diese Angabe weggelassen, wird vom aktuellen Roboter ausgegangen.

Rückgabewerte

WAHR, wenn der letzte Bewegungsbefehl erfolgreich beendet wurde, sonst FALSCH.

Verwandte Befehle

CurPos, FindPos

Beispiel einer InPos Funktion

```
Function main

  P0 = 0, -100, 0, 0
  P1 = 0, 100, 0, 0

  Xqt MonitorPosition, NoPause
  Do
    Jump P0
    Wait .5
    Jump P1
    Wait .5
  Loop

Fend

Function MonitorPosition

  Boolean oldInPos, pos

  Do
    Pos = InPos
    If pos <> oldInPos Then
      Print "InPos = ", pos
    EndIf
    oldInPos = pos
  Loop

Fend
```

Input Anweisung

> S

Gestattet es, numerische Daten von der Tastatur zu empfangen und in (einer) Variable(n) zu speichern.

Syntax

Input *varName* [*, varName, varName,...*]

Parameter

varName	Variablenname. Mit dem **Input**-Befehl können mehrfache Variablen verwendet werden, solange sie durch Kommata voneinader getrennt werden.

Beschreibung

Input empfängt numerische Daten vom Steuergerät und ordnet die Daten der/den Variable(n) zu, die mit dem **Input**-Befehl verwendet werden.

Wenn der Input-Befehl ausgeführt wird, erscheint ein (?)-Prompt am Steuergerät. Nach der Eingabe der Daten betätigen Sie bitte die Return/Enter-Taste auf der Tastatur.

Hinweise

Regeln für numerische Eingaben

Wenn numerische Werte eingegeben werden sollen und andere als numerische Werte eingegeben werden (außer dem Trennzeichen Komma), streicht die Input-Anweisung die nicht numerischen und alle folgenden Werte.

Regeln für Zeichenketten-Eingaben

Wenn Zeichenketten eingegeben werden, sind numerische- und Buchstabenzeichen als Daten gestattet.

Weitere Regeln für den Input-Befehl

- Wenn im Befehl mehr als eine Variable angegeben ist, müssen die Eingaben numerischer Daten für jede Variable durch Kommazeichen (",") voneinander getrennt werden.
- Numerische Variablennamen und Zeichenketten-Variablennamen sind gestattet. Der Eingabe-Datentyp muss jedoch dem Variablentyp entsprechen.

Potentielle Fehler

Die Anzahl der Variablen und der Eingabedaten sind unterschiedlich

Für Mehrfachvariablen muss die Anzahl der Eingabedaten mit der Anzahl der Eingabe-Variablennamen übereinstimmen. Wenn sich die Anzahl der Variablen, wie sie im Befehl definiert ist, unterscheidet von der Anzahl der von der Tastatur empfangenen numerischen Daten, tritt Fehler Nr. 30 auf.

Verwandte Befehle

Input #, Line Input, Line Input #, Print, String

Beispiel einer Input Anweisung

Diese Funktion zeigt einige einfache Beispiele einer Input-Anweisung.

```
Function InputNumbers
    Integer A, B, C

    Print "Please enter 1 number"
    Input A
    Print "Please enter 2 numbers separated by a comma"
    Input B, C
    Print "A = ", A
    Print "B = ", B, "C = ", C
Fend
```

```
Eine Mustersitzung des oben ablaufenden Programmes wird im Folgenden
dargestellt:
(Verwenden Sie das Run-Menü der die Taste F5, um das Programm zu
starten)

Please enter 1 number
?-10000
Please enter 2 numbers separated by a comma
?25.1, -99
-10000
25.1 -99
B = 25.1 C = -99
>
```

Input # AnweisungXXX

> S

Gestattet es, Zeichenketten- oder numerische Daten von einer Datei oder einer Kommunikationsschnittstelle zu empfangen und in (einer) Variable(n) zu speichern.

Syntax

Input #*handle, varName* [*, varName, varName,...*]

Parameter

handle	Die Datei- oder Kommunikationsport wird in einer Open-Anweisung definiert. Dateiports können in ROpen, WOpen, und AOpen Anweisungen definiert werden. Kommunikationsports können in OpenCom (RS232) und OpenNet (TCP/IP) Anweisungen definiert werden.
varName	Variablename, um Daten zu empfangen.

Beschreibung

Der **Input #** Befehl empfängt numerische- oder Zeichenkettendaten von dem Gerät, das mit *handle* gekennzeichnet ist, und ordnet die Daten der/den Variable(n) zu.

Hinweise

Regeln für numerische Eingaben

Wenn numerische Werte eingegeben werden sollen und andere als numerische Werte eingegeben werden (außer dem Trennzeichen Komma), streicht die Input-Anweisung die nicht numerischen und alle folgenden Werte.

Regeln für Zeichenketten-Eingaben

Wenn Zeichenketten eingegeben werden, sind numerische- und Buchstabenzeichen als Daten gestattet.

Weitere Regeln für den Input-Befehl

- Wenn im Befehl mehr als eine Variable angegeben ist, müssen die Eingaben numerischer Daten für jede Variable durch Kommazeichen (",") voneinander getrennt werden.
- Numerische Variablennamen und Zeichenketten-Variablennamen sind gestattet. Der Eingabe-Datentyp muss jedoch dem Variablentyp entsprechen.

Potentielle Fehler

Die Anzahl der Variablen und der Eingabedaten sind unterschiedlich

Wenn sich die Anzahl der Variablen, wie sie im Befehl definiert ist, unterscheidet von der Anzahl der vom Gerät empfangenen numerischen Daten, tritt Fehler Nr. 30 auf.

Verwandte Befehle

Input, Line Input, Line Input #

Beispiel einer Input # Anweisung

Diese Funktion zeigt einige einfache Beispiele einer Input # -Anweisung.

```
Function GetData
    Integer A
    String B$

    OpenCom #1
    Print #1, "Send"
    Input #1, A   'Nimmt numerische Daten vom Port #1 an
    Input #1, A   'Nimmt Zeichenketten-Daten vom Port #1 an
    CloseCom #1
Fend
```

InputBox Anweisung

> S

Zeigt ein Prompt in einer Dialogbox an, wartet darauf, dass der Anwender Text eingibt oder einen Button anwählt und gibt den Inhalt der Box aus.

Syntax

InputBox *prompt, title, default, answer$*

Parameter

prompt	Zeichenkettenausdruck, der als Nachricht im Dialogfeld angezeigt wird.
title	Zeichenkettenausdruck, der in der Titelzeile des Dialogfensters angezeigt wird.
default	Zeichenkettenausdruck, der im Textfenster als Vorgabeantwort angezeigt wird. Wenn keine Vorgabe gewünscht wird, ist eine leere Zeichenkette ("") zu verwenden.
answer$	Eine Zeichenkettenvariable, die enthält, was der Anwender eingibt. Wenn der Anwender nicht eingibt oder auf Cancel klickt, so lautet diese Zeichenkette "@".

Beschreibung

InputBox zeigt den Dialog an und wartet darauf, dass der Anwender auf OK oder Cancel klickt. *answer* ist eine Zeichenkette, die enthält, was der Anwender eingegeben hat.

Verwandte Befehle

MsgBox

Beispiel einer InputBox Anweisung

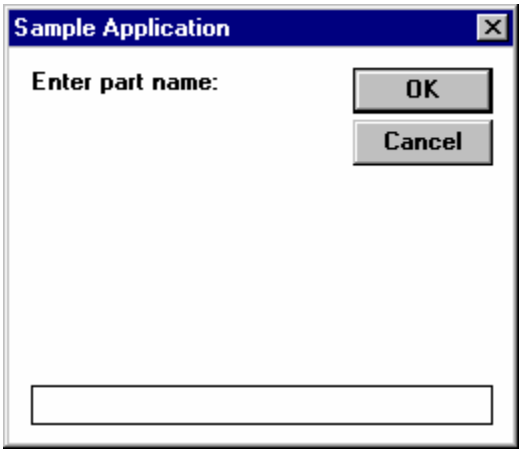
Diese Funktion zeigt ein InputBox Beispiel.

```
Function GetPName
    String tprompt$, def$, title$, answer$

    tprompt$ = "Enter part name:"
    title$ = "Sample Application"
    def$ = NamPart$
    InputBox tprompt$, title$, def$, answer$
    If answer$ <> "@" Then
        NamPart$ = answer$
    EndIf
Fend
```

Beispiel-Ausgabe

Das folgende Bild zeigt ein Beispiel der Ausgabe des InputBox Beispiel-Codes, wie oben gezeigt.



InStr Funktion

F

Gibt die Position einer Zeichenkette innerhalb einer anderen aus.

Syntax

InStr(*string, searchString*)

Parameter

string	Zeichenkettenausdruck, in dem gesucht werden soll.
searchString	Zeichenkettenausdruck, nach dem innerhalb von *string* gesucht werden soll.

Rückgabewerte

Gibt die Position der Suchkette aus, wenn der Ort gefunden wird, sonst wird -1 ausgegeben.

Verwandte Befehle

Mid$

Beispiel einer Instr Funktion

```
Integer pos

pos = InStr("abc", "b")
```

Int Funktion

F

Konvertiert eine Real-Zahl in eine Integer-Zahl. Gibt den größten Integer aus, der kleiner oder gleich dem spezifizierten Wert ist.

Syntax

Int*(number)*

Parameter

number Ein Ausdruck mit einer Real-Zahl.

Rückgabewerte

Gibt einen Integer-Wert der in *number* verwendeten Real-Zahl aus.

Beschreibung

Int(*number*) nimmt den Wert von *number* und gibt den größten Integer aus, der kleiner oder gleich *number* ist.

Hinweis

Für Werte kleiner als 1 (negative Zahlen)

Wenn der Parameter *number* einen Wert von weniger als 1 hat, dann hat der Rückgabewert einen größeren Absolutwert, als *number*. (Wenn die Zahl z.B. folgendermaßen lautet: -1.35 so wird -2 ausgegeben.)

Verwandte Befehle

Abs, Atan, Atan2, Cos, Mod, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Beispiel einer Int Funktion

Einige einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> Print Int(5.1)
5
> Print Int(0.2)
0
> Print Int(-5.1)
-6
>
```

Integer Anweisung

S

Deklariert Integer-Variablen. (2 Byte Integer).

Syntax

Integer *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]**...**

Parameter

varName	Variablenname, den der Anwender als Integer deklarieren will.
subscripts	Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt **(***dim1* [, *dim2* [, *dim3*]]**)** *dim1*, *dim2*, *dim3* können eine Integer-Zahl zwischen 0-32767 sein.

Beschreibung

Integer wird verwendet, um Variablen als **Integer** zu deklarieren. Integer-Variablen können ganze Zahlen im Wertebereich von -32768 bis 32767 enthalten. Alle lokalen Variablen sollten in einer Funktion ganz oben deklariert sein.

Verwandte Befehle

Boolean, Byte, Double, Global, Long, Real, String

Beispiel einer Integer Anweisung

Das folgende Beispiel zeigt ein einfaches Programm, das unter Verwendung von **Integer** einige Variablen deklariert.

```
Function inttest
    Integer A(10)         'Eindimensionale Feldvariable aus Integers
    Integer B(10, 10)     'Zweidimensionale Feldvariable aus Integers
    Integer C(10, 10, 10) 'Dreidimensionale Feldvariable aus Integers
    i
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For I = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next I
Fend
```

InW Funktion

F

Gibt den Status des spezifizierten Eingangs-Wordport aus. Jeder Wordport enthält 16 Eingabebits.

Syntax

InW(*number*)

Parameter

number Spezifiziert die Wordportnummern innerhalb des angegebenen I/O-Platinen-Bereichs unter Verwendung eines Integer-Ausdrucks.

Rückgabewerte

Gibt den aktuellen Eingabe-Status aus (Long Integers von 0 bis 65535).

Verwandte Befehle

In, Out, OutW

Beispiel einer InW Funktion

```
Long word0

word0 = InW(0)
```

JA Funktion

F

Gibt einen in Achswinkeln spezifizierten Roboterpunkt aus.

Syntax

JA(*j1*, *j2*, *j3*, *j4***)**

Parameter

j1 - j4 Real-Ausdrücke, welche Achswinkel repräsentieren.

Rückgabewerte

Ein Roboterpunkt, dessen Ort durch die dargestellten Achswinkel bestimmt wird.

Verwandte Befehle

XY

Beispiel einer JA Funktion

```
P10 = JA(60,30,-50,45)
GO JA(135,90,-50,90)
```

JRange Anweisung

> S

Definiert den erlaubten Arbeitsbereich der spezifizierten Achse in Pulsen.

Syntax

JRange *jointNumber, lowerLimit, upperLmit*

Parameter

jointNumber	Integer-Ausdruck zwischen 1 und 4, der die Achse repräsentiert, für das JRange spezifiziert wird.
lowerLmit	Long Integer-Ausdruck, der den Zählstand der die Encoder-Pulse-Position für den Bereich des unteren Grenzwerts der angegebenen Achse repräsentiert.
upperLmit	Long Integer-Ausdruck, der den Zählstand der die Encoder-Pulse-Position für den Bereich des oberen Grenzwerts der angegebenen Achse repräsentiert.

Beschreibung

Definiert den erlaubten Arbeitsbereich für die definierte Achse mit unterem und oberem Grenzpunkt in Encoder-Pulsen. **JRange** ist dem Range-Befehl ähnlich. Jedoch müssen für den Range-Befehl alle Grenzwerte der 4 Achsen angegeben werden, während der **JRange**-Befehl verwendet werden kann, um den Grenzwert jeder Achse individuell einzustellen, was wiederum die Anzahl der benötigten Parameter reduziert. Um den definierten Arbeitsbereich zu überprüfen, verwenden Sie bitte den Range-Befehl.

Hinweise

Die unteren Grenzwerte dürfen die oberen Grenzwerte nicht überschreiten:

Der untere Grenzwert, der im JRange-Befehl definiert ist, darf den oberen Grenzwert nicht überschreiten.Wenn der untere Grenzwert den oberen Grenzwert übersteigt, ruft dies einen Fehler hervor.

Faktoren, die JRange ändern können:

Sobald **JRange**-Werte eingestellt sind, bleiben sie bestehen, bis der Anwender diese Werte entweder durch den Range- oder den **JRange**-Befehl ändert. Weder das Ausschalten der Versorgungsspannung, noch die Ausführung des Verinit-Befehls ändert die Werte der **JRange** Achs-Grenzwerte.

Maximaler und minimaler Abeitsbereich:

Lesen Sie bitte die Spezifikationen im Roboterhandbuch bezüglich der maximalen Arbeitsbereiche für jedes Robotermodell, da diese Werte sich von Modell zu Modell unterscheiden.

Verwandte Befehle

Range, JRange Funktion

Beispiel einer JRange-Anweisung

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt:

```
> JRange 2, -6000, 7000          Definiert den zweiten Armbereich

> JRange 1, 0, 7000          'Definiert den ersten Armbereich
```

JRange Funktion

F

Syntax

JRange(*jointNumber, paramNumber*)

Parameter

jointNumber	Spezifiziert die Referenz-Achsnummer (Integer von 1 bis 6) durch einen Ausdruck oder einen numerischen Wert.
paramNumber	Integer-Ausdruck, der einen von zwei Werten enthält: 1: Spezifiziert den Wert der unteren Grenze. 2:Spezifiziert den Wert der oberen Grenze.

Rückgabewerte

Bereichseinstellung (Integer-Wert, Pulses) der spezifizierten Achse.

Verwandte Befehle

Range, JRange Anweisung

Beispiel einer JRange Funktion

```
Long i, oldRanges(3, 1)

For i = 0 To 3
    oldRanges(i, 0) = JRange(i + 1, 1)
    oldRanges(i, 1) = JRange(i + 1, 2)
Next i
```

JS Funktion

F

Jump Sense ermittelt, ob der Arm angehalten hat, bevor ein Jump-Befehl (der einen Sense-Eingang verwendet hat) vollständig ausgeführt wurde, oder ob der Arm die Jump-Bewegung ganz ausgeführt hat.

Syntax

JS

Rückgabewerte

Gibt eine 1 oder eine 0 aus.

- **1:** Wenn der Arm gestoppt wurde, bevor er seinen Zielpunkt erreicht hat, weil eine Sense-Eingangs-Bedingung erfüllt wurde, gibt **JS** eine 1 aus.
- **0:** Wenn der Arm die normale Bewegung abschließt, und den durch den Jump-Befehl definierten Zielort erreicht, gibt **JS** eine 0 aus.

Beschreibung

JS wird zusammen mit den Befehlen Jump und Sense verwendet. Der Zweck des **JS-**Befehls ist es**,** ein Statusergebnis zur Verfügung zu stellen, welches aussagt, ob ein eine Eingabebedingung (die durch den Sense-Befehl definiert wurde) während einer Bewegung (die durch den Jump-Befehl ausgelöst wurde) erfüllt wird, oder nicht. Wenn die Eingabebedingung erfüllt wird, gibt **JS** eine 1 aus. Wenn die Eingabebedingung nicht erfüllt wird und der Arm die Zielposition erriecht, gibt **JS** eine 0 aus.

JS ist einfach ein Befehl zur Status-Überprüfung, verursacht keinerlei Bewegung oder spezifiziert, welcher Eingang während der Bewegung überprüft werden soll. Der Jump-Befehl wird verwendet, um Bewegungen zu initiieren und der Sense-Befehl wird verwendet, um zu spezifizieren, welcher Eingang während einer durch den Jump-Befehl ausgelösten Bewegung überprüft werden soll (so überhaupt einer überprüft werden soll).

HINWEIS

JS funktioniert nur mit dem zuletzt verwendeten Jump-Befehl:

JS kann nur verwendet werden, um den **Eingangs-Check** des zuletzt verwendeten Jump-Befehls zu überprüfen (welcher durch den Sense-Befehl initiiert wurde.) Sobald ein zweiter Jump-Befehl initiiert wird, kann **JS** nur den Status für den zweiten Jump-Befehl ausgeben. Der **JS-**Status für den ersten Jump-Befehl ist nicht mehr verfügbar. Stellen Sie sicher, dass Sie ihre **JS**-Status-Checks für Jump-Befehle immer direkt nach einem ausgeführten, zu prüfenden Jump-Befehl tätigen.

Verwandte Befehle

JT, Jump, Sense

Beispiel einer JS Funktion

```
Function SearchSensor As Boolean
    Sense Sw(5) = 1

    Jump P0
    Jump P1 Sense
    If JS = 1 Then
        Print "Sensor was found"
        SearchSensor = TRUE
    EndIf
Fend
```

JT Funktion

F

Gibt den Status des zuletzt ausgegebenen JUMP-Befehls für den aktuellen Roboter aus.

Syntax

JT

Rückgabewerte

JT gibt einen Long-Wert aus, angegeben durch die folgenden Bits:

Bit 0	Auf 1 gesetzt, wenn die Hubbewegung begonnen hat oder die Hubdistanz 0 beträgt.
Bit 1	Auf 1 gesetzt, wenn die Horizontalbewegung begonnen hat oder die Horizontaldistanz 0 beträgt.
Bit 2	Auf 1 gesetzt, wenn die Senkbewegung begonnen hat oder die Senkdistanz 0 beträgt.
Bit 16	Auf 1 gesetzt, wenn die Hubbewegung abgeschlossen ist oder die Hubdistanz 0 beträgt.
Bit 17	Auf 1 gesetzt, wenn die Horizontalbewegung abgeschlossen ist oder die Horizontaldistanz 0 beträgt.
Bit 18	Auf 1 gesetzt, wenn die Senkbewegung abgeschlossen ist oder die Senkdistanz 0 beträgt.

Beschreibung

JT wird verwendet, um den Status des zuletzt ausgeführten Jump-Befehls festzustellen, der vor seiner Vervollständigung durch einen der Befehle Sense, Till, Abort, etc gestoppt wurde.

Verwandte Befehle

JS, Jump, Sense, Till

Beispiel einer JT Funktion

```
Function SearchTill As Boolean

    Till Sw(5) = On

    Jump P0
    Jump P1 Till
    If JT And 4 Then
        Print "Motion stopped during descent"
        SearchTill = TRUE
    EndIf
Fend
```

JTran Anweisung

S

Führt eine relative Bewegung einer Achse aus.

Syntax

JTran *jointNumber, distance*

Parameter

jointNumber	Integer-Ausdruck, der darstellt, welche Achse zu bewegen ist.
distance	Real-Ausdruck, der die Distanz repräsentiert, die zurückgelegt werden muss, für Rotationsachsen in Gradangaben und für Linearachsen in Millimetern.

Beschreibung

JTran wird verwendet, um eine Achse über eine derfinierte Distanz von seiner aktuellen Position weg zu bewegen.

Verwandte Befehle

Go, Jump, Move, PTran

Beispiel einer JTran Anweisung

```
JTran 1, 20
```

Jump Anweisung

> S

Bewegt den Arm mit einer PTP-Bewegung von seiner aktuellen Position zum angegebenen Zielpunkt, indem der Arm zunächst vertikal nach oben fährt, dann horizontal und zum Schluss vertikal abwärts, um an den Zielpunkt der Bewegung zu gelangen.

Syntax

Jump *destination* [**C***archNumber*] [**LimZ** *zLimit*] [*searchExpr*] [*!...!*]

Parameter

destination	Der Zielort einer Bewegung, die einen Punktausdruck verwendet.
archNumber	Optional. Die Arch-Nummer (*archNumber*) definiert, welcher Eintrag der Arch-Tabelle für die Arch-Bewegung verwendet werden soll, die durch den Jump-Befehl hervorgerufen wird. Der *archNumber* muss immer der Buchstabe C vorangestellt werden. (Gültige Einträge sind C0-C7.)
zLimit	Optional. Dies ist ein Z-Grenzwert, der die Maximalposition repräsentiert, zu der sich die Z-Achse während einer **Jump**-Bewegung bewegen lässt. Man kann dies als Z-Höhen-Obergrenze für den **Jump**-Befehl betrachten. Jeglicher gültige Z-Achsen-Koordinatenwert ist zulässig.
searchExpr	Optional. Ein Sense-, Till- oder Find- Ausdruck. **Sense** | **Till** | **Find** **Sense Sw**(*expr*) = {**Ein** | **Aus**} **Till Sw**(*expr*) = {**Ein** | **Aus**} **Find Sw**(*expr*) = {**Ein** | **Aus**}
!...!	Optional. Parallelbearbeitungsanweisungen können zum Jump-Befehl hinzugefügt werden, um I/O- und andere Befehle während der Bewegung auszuführen.

Beschreibung

Jump bewegt den Arm von seiner aktuellen Position zur *destination* unter Verwendung der so genannten Arch-Bewegung (Bogen-Bewegung). Man kann Jump als drei Bewegungen in einer betrachten. Wenn beispielsweise der durch *archNumber* definierte Arch-Tabellen-Eintrag 5 ist, werden die folgenden drei Bewegungen ausgeführt.

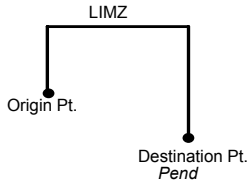
1) Die Bewegung beginnt nur mit der Z-Achse, bis diese die Höhe erreicht hat, die durch die Arch-Nummer errechnet wurde, welche für den Jump-Befehl verwendet wird.
2) Als nächstes bewegt der Arm sich horizontal (während er in der Z-Achse noch immer die Hubbewegung durchführt) in Richtung der Zielpunkt-Position, bis der obere Grenzpunkt der Z-Achse (definiert durch LimZ) erreicht ist. Dann beginnt der Arm, sich in Z-Richtung abwärts zu bewegen (während er mit den X, Y und U Achsen-Bewegungen fortfährt), bis die endgültigen X, Y und U Achsen-Positionen erreicht sind.
3) Der **Jump**-Befehl wird dann abgeschlossen, indem der Arm nur mit einer Bewegung der Z-Achse abwärts bewegt wird, bis die Zielposition der Z-Achse erreicht ist.

Die *destination* Koordinaten (Die Zielposition für die Bewegung) müssen vor dem Ausführen des **Jump** Befehls eingeteacht werden. Die Koordinaten können nicht im **Jump**–Befehl selber spezifiziert werden. Beschleunigung und Verzögerung für die **Jump**-Anweisung werden durch den Accel-Befehl gesteuert. Die Geschwindigkeit für die Bewegung wird durch den Speed-Befehl gesteuert.

***archnum* Details**

Der Bogen für die **Jump**-Anweisung kann, auf der Basis des *archNumber*-Wertes, der optional zusammen mit dem Jump-Befehl definiert wird, verändert werden. Dies gestattet es dem Anwender, zu definieren wie viel der Z-Bewegung zurückgelegt werden soll, bevor mit der Bewegung der X, Y, und U –Achse begonnen wird. (Der Anwender kann so den Arm aufwärts bewegen und aus dem Weg von Teilen, Feedern und anderen Gegenständen halten, bevor mit der Horizontalbewegung begonnen wird.) Gültige *archNumber* –Einträge für den **Jump-**Befehl bewegen sich zwischen C0 und C7. Die Arch-Tabellen-Einträge für C0-C6 können über den Arch-Befehl vom Anwender definiert werden. C7 ist jedoch ein besonderer Arch-Eintrag, der immer die so genannte Gate-Bewegung definiert. Gate-Bewegung bedeutet, dass der Roboter die Z-Achse zuerst vollständig zu der durch LimZ definierten

Koordinate bewegt, bevor die X, Y, oder U –Achsen-Bewegungen begonnen werden. Sobald der LimZ Z-Grenzwert erreicht ist, beginnt die Bewegung der Achsen X, Y und U. Nachdem die Achsen X, Y, und U jeweils ihre Zielposition erreicht haben, kann die Z-Achse mit ihrer Abwärtsbewegung zur letzten Z-Achsen-Koordinatenposition beginnen, die durch *destination* (den Zielpunkt) definiert wurde. Die Gate-Bewegung sieht aus wie folgt:



LimZ Details
LimZ zCoord spezifiziert den oberen Z-Koordinatenwert für die Horizontalbewegungs-Ebene. Die Definition von *archnum* kann die Achsen X, Y, und U dazu veranlassen, mit ihrer Bewegung zu beginnen, bevor LimZ erreicht ist, aber LimZ ist immer die maximale Z-Höhe für die Bewegung. Wenn der optionale LimZ-Parameter weggelassen wird, wird der davor durch den LimZ-Befehl spezifizierte Wert für die Definition der Horizontalbewegungsebene verwendet.

Es ist wichtig zu beachten, dass die Spezifikation der LimZ *zLimit* -Höhenbegrenzung den Z-Achsen-Wert für das Roboterkoordinatensystem darstellt. Es ist nicht der Z-Achsen-Wert für Arm, Tool, oder LOCAL0 Koordinaten. Treffen Sie daher die notwendigen Schutzvorkehrungen für den Gebrauch von Werkzeugen oder Armen mit unterschiedlicher Betriebshöhe.

Sense Details
Der optionale Sense-Parameter gestattet es dem Anwender, eine Eingangsbedingung zu überprüfen (oder S/W Speicher I/O Bedingung) bevor die endgültige Abwärtsbewegung der Z-Achse beginnt. Ist sie erfüllt, dann wird dieser Befehl beendet und der Roboter stoppt über der Zielposition (von wo nur die Bewegung der Z-Achse benötigt wird, um die Zielposition zu erreichen). Es ist wichtig zu beachten, dass der Roboterarm bei Wahrnehmen der Sense-Eingangsbedingung nicht sofort anhält.

Die Befehle Js oder Stat können dann verwendet werden, um zu beurteilen, ob die Sense-Bedingung erfüllt wurde und der Roboter vor seiner Zielposition gestoppt hat oder ob die Sense-Bedingung nicht erfüllt wurde und der Roboter seinen Weg bis zur Zielposition fortgesetzt hat.

Till Details
Die optionale Till-Bedingung gestattet es dem Anwender eine Bedingung zu definieren, die den Roboter dazu veranlasst, zu bis zum vollständigen Stopp zu verzögern, bevor der **Jump**-Befehl vollständig ausgeführt wurde. Die definierte Bedingung ist einfach nur ein Gegenprüfung zu einem der 512 Eingänge oder einem der S/W Merker. Dies wird durch Verwenden der Funktionen Sw oder Sw($) erreicht. Der Anwender kann überprüfen, ob die Eingänge aus- oder eingeschaltet sind und den Arm dazu veranlassen, zu verzögern und anzuhalten, je nachdem, welche Bedingung spezifiziert ist. Es ist wichtig zu beachten, dass der Robter nicht umgehend stoppt, sobald die Till-Bedingung erfüllt ist. Tatsächlich stoppt der Roboter auf der definierten LimZ-Höhe. Wenn LimZ nicht definiert ist, bewegt sich der Roboter zu der Höhenposition, in der Z=0 ist. Wenn die Bedingung während des **Jump**-Befehls nie erfüllt wird, erreicht der Arm erfolgreich den durch *destination* definierten Punkt.

Es ist wichtig zu beachten, dass die Till-Bedingung während der Roboterbewegung überprüft wird, bis die Z-Achse (Achse 3) die Abwärtsbewegung beginnt. Sobald die Z-Achse (Achse 3) die Abwärtsbewegung beginnt, wird die Till-Bedingung nicht mehr geprüft.

Die Stat-Funktion kann verwendet werden, um zu verifizieren, ob die Till-Bedingung erfüllt wurde und dieser Befehl vollständig ausgeführt wurde, oder ob die Till-Bedingung nicht erfüllt wurde und der Roboter an der Zielposition angehalten hat.

Positionstabelle für Till:
Im Folgenden sehen Sie die Positionstabelle, welche mehrere Fälle auflistet, für die dieTill-Bedingung erfüllt wurde, und die entsprechende Stopp-Position des Roboters.

Position: Bedingung erfüllt	Position: Bewegung gestoppt
1) Während der vertikalen Hubbewegung	Der Arm fährt bis zur LimZ-Position hoch und verzögert dann bis zum vollständigen Stopp.
2) Während der Bewegungskombination (horizontal und vertikal)	Die Horizontalbewegung verzögert und stoppt. Die Vertikalbewegung geht aufwärts und stoppt.
3) Während der Horizontalbewegung, in der LimZ Z-Position (D.h. keine Vertikalbewegung)	Der Arm verzögert und stoppt.
4) Nachdem der Arm sich in Z abwärts bewegt.	Der Arm stoppt an seiner Zielposition.

Hinweise

Auslassen des *archNumber* Parameters

Wenn der optionale *archnum*-Parameter weggelassen wird, lautet der Vorgabe-Arch-Eintrag für die Verwendung mit dem Jump-Befehl C7. Dies verursacht die Gate-Bewegung, wie oben beschrieben.

Der Unterschied zwischen Jump und Go:

Der Go-Befehl ist **Jump** in sofern ähnlich, als beide PTP-Bewegungen verursachen. Dabei gibt es jedoch viele Unterschiede. Der wichtigste Unterschied ist, dass der Go-Befehl PTP-Bewegungen verursacht, bei denen alle Achsen zur selben Zeit starten und anhalten (sie sind synchronisiert). Der Jump-Befehl unterscheidet sich davon, indem er die vertikale Z-Bewegung zu Beginn und Ende der Bewegung ausführt. Der Jump-Befehl ist ideal für Bestückungs-Anwendungen.

Verzögerung bis zum vollständigen Halt mit dem Jump-Befehl

Korrekte Geschwindigkeits- und Beschleunigungs-Befehle mit Jump:

Die Befehle Speed und Accel werden verwendet, um Geschwindigkeit und Beschleunigung des Roboters während der **Jump-**Bewegung zu definieren. Achten Sie darauf, dass die Befehle Speed und Accel sich auf PTP-Bewegungen beziehen (Go, Jump, etc), während für linearinterpolierte und kreisinterpolierte Bewegungen (wie Move oder Arc) die Befehle SpeedS und AccelS verwendet werden. Für den Jump-Befehl ist es möglich, Geschwindigkeit und Beschleunigung für die folgenden Bewegungen separat zu definieren: Hubbewegung der Z-Achse, horizontale Verfahrbewegung inklusive U-Achsen-Rotation und die Abwärtsbewegung der Z-Achse.

Potentielle Fehler

Der LimZ-Wert ist nicht hoch genug

Wenn die aktuelle Armposition der Z-Achse höher ist als der Wert, der für LimZ eingestellt wurde und der Versuch unternommen wird, einen Jump-Befehl auszuführen, tritt Fehler Nr. 146 auf.

Verwandte Befehle

Accel, Arc, Arch, Go, Js, Jt, LimZ, Point Expression (Punktausdruck), Pulse, Sense, Speed, Stat, Till

Beispiel einer Jump Anweisung

Das folgende Beispiel zeigt eine einfache PTP-Bewegung zwischen den Punkten P0 und P1 und die darauf folgende Rückkehr zum Punkt P0 unter Verwendung des Jump-Befehls. Später im Programm bewegt sich der Arm vermittels des Jump-Befehls, bis am Eingang #4 Spannung anliegt. Wenn an Eingang #4 Spannung anliegt, stoppt der Arm seine Bewegung. Wenn am Eingang #4 nie Spannung anliegt, führt der Arm den Jump-Befehl vollständig aus und kommt bei Punkt P1 an.

```
Function jumptest
    Home
    Go P0
    Go P1
    Sense Sw(4) = 1
    Jump P0 LimZ -10
    Jump P1 LimZ-10 Sense   'Überprüft den Eingang #4
    If Js(0) = 1 Then
        Print "Input #4 came on during the move and"
        Print "the robot stopped prior to arriving on"
        Print "point P1."
    Else
        Print "The move to P1 completed successfully."
        Print "Input #4 never came on during the move."
    EndIf
Fend
```

<Andere Beispielsyntax vom Online-Fenster>

```
> Jump P10+X50 C0 LimZ-20 Sense !D50;On 0;D80;On 1!
```

Kill Anweisung

>

Löscht Dateien auf einem Laufwerk des PCs.

Syntax

Kill *pathName*

Parameter

pathName Zeichenkettenausdruck, der den zu löschenden Dateinamen enthält. *pathName* kann das Verzeichnis oder den Ordner und das Laufwerk enthalten.

Verwandte Befehle

Del, Dir, SavePoints

Biepiel einer Kill Anweisung

Das folgende Beispiel wird vom Online-Fenster aus ausgeführt:

```
>Kill "ABCD.DAT"
>
```

LCase$ Funktion

F

Gibt eine Zeichenkette aus, die in Kleinbuchstaben umgewandelt wurde.

Syntax

LCase$(*string*)

Parameter

string Ein gültiger Zeichenkettenausdruck.

Rückgabewerte

Die konvertierte Kleinbuchstaben-Zeichenkette.

Verwandte Befehle

UCase$

Beispiel einer LCase$ Funktion

```
str$ = "Data"
str$ = LCase$(str$)  ' str$ = "data"
```

Left$ Funktion

F

Gibt eine Teilkette von der linken Seite eines Zeichenkettenausdrucks aus.

SyntaxLeft$(string, count)

Parameter

string	Zeichenkettenausdruck, von dem die äußerst links gelegenen Zeichen kopiert werden.
count	Anzahl der zu kopierenden Zeichen aus *string,* beginnend mit dem Zeichen, das äußerst links steht.

Rückgabewerte

Gibt eine Zeichenkette der äußerst links stehenden Zeichen aus der durch den Anwender definierten Buchstabenzeichenkette aus.

Beschreibung

Left$** gibt die äußerst links gelegenen Zeichen einer durch den Anwender definierten Zeichenkette aus. **Left$ kann so viele Zeichen ausgeben, wie in der Zeichenkette vorhanden sind.

Verwandte Befehle

Asc, Chr$, Len, Mid$, Right$, Space$, Str$, Val

Beispiel einer Left$ Funktion

Das Beispiel unten zeigt ein Programm, das Daten eines Teils als Zeichenkette übergeben bekommt und diese in Teilnummer, Teilnamen und Teileanzahl zerlegt.

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(datain$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(datain$, Len(DataIn$) - pos))

Fend
```

Einige weitere Beispiele des Left$-Befehls vom Online-Fenster.

```
> Print Left$("ABCDEFG", 2)
 AB

> Print Left$("ABC", 3)
 ABC
```

Len Funktion

F

Gibt die Anzahl von Zeichen in einer Zeichenkette aus.

Syntax

Len*(string)*

Parameter

string Zeichenkettenausdruck.

Rückgabewerte

Gibt eine Integer-Zahl aus, welche die Anzahl der Zeichen in der *string*-Zeichenkette darstellt, die als Argument an den **Len**-Befehl gegeben wurde.

Beschreibung

Len gibt einen Integer aus, welcher die Anzahl von Zeichen in einer Zeichenkette repräsentiert, die durch den Anwender definiert wurde. **Len** gibt Werte zwischen 0 und 256 aus (da eine Zeichenkette zwischen 0 und 256 Zeichen enthalten kann).

Verwandte Befehle

Asc, Chr$, Left$, Mid$, Right$, Space$, Str$, Val

Beispiel einer Len Funktion

Das Beispiel unten zeigt ein Programm, das Daten eines Teils als Zeichenkette übergeben bekommt und diese in Teilnummer, Teilnamen und Teileanzahl zerlegt.

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(datain$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(datain$, Len(DataIn$) - pos))

Fend
```

Einige weitere Beispiele des **Len**-Befehls vom Online-Fenster.

```
> ? len("ABCDEFG")
7

> ? len("ABC")
3

> ? len("")
0
>
```

LimZ Anweisung

> S

Bestimmt den Vorgabewert für die Höhe der Z-Achse bei Jump-Befehlen.

Syntax

LimZ [*zLimit*]

Parameter

zLimit Ein Koordinatenwert innerhalb des beweglichen Bereichs der Z-Achse.

Rückgabewerte

Zeigt den aktuellen LimZ-Wert an, wenn der Parameter weggelassen wird.

Beschreibung

LimZ bestimmt die maximale Höhe, die die Z-Achse erreicht, wenn der Jump-Befehl verwendet wird, wobei der Roboterarm die Z-Achse hebt, sich in der X-Y-Ebene bewegt, und die Z-Achse wieder senkt. Einfach ausgedrückt ist **LimZ** der Vorgabewert für die Z-Achse, der verwendet wird, um die höchste zu verwendende Position zu definieren, die bei einer durch den Jump-Befehl ausgelösten Bewegung verwendet wird. Wenn ein spezifischer **LimZ-**Wert im Jump-Befehl nicht definiert ist, wird die letzte **LimZ** Einstellung für den Jump-Befehl verwendet.

Hinweis

Zurücksetzen des LimZ-Wertes auf 0

Ein Neustart der Software und die Ausführung der Befehle SFree, SLock, Motor On und Verinit setzen den LimZ-Wert auf 0 zurück.

Der LimZ-Wert ist nicht gültig für Arm-, Werkzeug- oder Lokale Koordinaten:

Die Höhenbegrenzung für die Z-Achse entspricht dem Z-Achsen-Wert für das Roboter-Koordinatensystem. Es ist nicht der Z-Achsen-Wert für Arm, Tool oder lokale Koordinaten. Treffen Sie daher die notwendigen Schutzvorkehrungen für den Gebrauch von Werkzeugen oder Greifern mit unterschiedlicher Betriebshöhe.

Verwandte Befehle

Jump

Beispiel einer LimZ Anweisung

Das Beipiel unten zeigt die Verwendung von LimZ in Jump-Operationen.

```
Function main
    LimZ -10            'Setzt den LimZ-Standardwert
    Jump P1             'Bewegt die Achse für den Jump-Befehl aufwärts
                         in die Position Z=-10
    Jump P2 LimZ -20    'Bewegt die Achse für den Jump-Befehl aufwärts
                         in die Position Z=-20
    Jump P3             'Bewegt die Achse für den Jump-Befehl aufwärts
                         in die Position Z=-10
Fend
```

LimZ Funktion

F

Gibt die aktuelle LimZ Einstellung aus.

Syntax

LimZ

Rückgabewerte

Real-Zahl, welche die aktuelle LimZ-Einstellung enthält.

Verwandte Befehle

LimZ Anweisung

Beispiel einer LimZ Funktion

```
Real savLimz

savLimz = LimZ
LimZ -25
Go pick
LimZ savLimZ
```

Line Input Anweisung

> S

Liest eine Zeile der Eingangsdaten und ordnet die Daten einer Zeichenkettenvariable zu.

Syntax-{}-

Line Input *stringVar$*

Parameter

stringVar$	Name einer Zeichenkettenvariable. (Bedenken Sie, dass die Zeichenkettenvariable mit dem $-Zeichen enden muss.)

Beschreibung

Line Input liest eine Zeile der Eingangsdaten des Steuerungsgerätes und ordnet die Daten der Zeichenkettenvariable zu, die in der Line Input Anweisung verwendet wird. Wenn die Line Input Anweisung bereit ist, Daten vom Anwender zu empfangen, wird ein "?" -Prompt auf dem Steuerungsgerät angezeigt. Die Eingangsdatenzeile nach dem Prompt wird dann als Wert für die Zeichenkettenvariable aufgenommen. Nach Eingabe der Datenzeile drücken Sie bitte die [ENTER]-Taste.

Verwandte Befehle

Input, Input #, Line Input#, ParseStr

Beispiel einer Line Input Anweisung

Das folgende Beispiel zeigt die Verwedung von **Line Input**.

```
Function Main
    String A$
    Line Input A$  ' Liest eine Zeile der Eingangsdaten in A$ ein
    Print A$
Fend
```

```
Starten Sie das obige Programm mit der F5-Taste oder aus dem Run-Menü
des EPSON RC+-Hauptfensters. Eine sich ergebende Run-Session kann wie
folgt aussehen:

?A, B, C
A, B, C
```

Line Input # Anweisung

> S

Liest die Daten einer Zeile von der angegebenen Datei oder der Kommunikationsschnittstelle.

Syntax

Line Input #*portNumber, stringVar$*

Parameter

portNumber	Integer-Ausdruck, der eine Dateinummer oder die Nummer einer Kommunikationsschnittstelle repäsentiert.
stringVar$	Eine Zeichenkettenvariable. (Bedenken Sie, dass Zeichenkettenvariablen mit dem $-Zeichen enden müssen.)

Beschreibung

Line Input # liest Zeichenkettendaten einer Zeile aus der Datei oder der Kommunikationsschnittstelle, die durch den Parameter *portNumber* spezifiziert ist, und ordnet die Daten der Zeichenkettenvariable *stringVar$* zu.

Verwandte Befehle

Input, Input #, Line Input

Beispiel einer Line Input # Anweisung

In diesem Beispiel werden die Zeichenkettendaten von Kommunikationsschnittstelle Nr. 1 empfangen und der Zeichenkettenvariablen and A$ zugeordnet.

```
Function lintest
    String a$
    Print #1, "Please input string to be sent to robot"
    Line Input #1, a$
    Print "Value entered = ", a$
Fend
```

LoadPoints Anweisung

> S

Lädt eine Punktdatei in den Punktspeicherbereich für den aktuellen Roboter.

Syntax

LoadPoints *fileName* **[, Merge]**

Parameter

fileName	Zeichenkettenausdruck, der den spezifischen Pfad und die Datei enthält, die in den Punktspeicherbereich des aktuellen Roboters geladen werden sollen. Die Dateinamenerweiterung muss .PNT lauten.
Merge	Optional. Wenn Merge angegeben wird, werden die aktuellen Punkte nicht gelöscht, bevor die neuen Punkte geladen werden. Die Punkte in der Datei werden den aktuellen Punkten hinzugefügt. Wenn ein Punkt in der Datei bereits existiert, wird der im Speicher befindliche Punkt überschrieben.

Beschreibung

LoadPoints lädt Punktdateien von einem Laufwerk in den Hauptspeicherbereich der Steuerung. Verwenden Sie **Merge,** um Punktdateien zu kombinieren. So können Sie beispielsweise eine Haupt-Punktdatei haben, welche die gemeinsamen Punkte für Locals, Parken etc. im Bereich von 0 – 100 enthält. **Merge** wird dann verwendet, um andere Punktdateien für jedes verwendete Teil zu laden, ohne die gemeinsam verwendeten Punkte zu löschen. Der Bereich kann z.B. zwischen 101 – 999 liegen.

Potentielle Fehler

Die Datei existiert nicht:

Wenn die in der **LoadPoints** Anweisung definierte Datei nicht existiert, wird Fehler Nr. 53 ausgegeben.

Die Punktedatei ist nicht für den aktuellen Roboter bestimmt.

Wenn es sich bei der in der **LoadPoints** Anweisung definierten Datei nicht um eine Punktdatei für den aktuellen Roboter handelt, wird die folgende Fehlermeldung ausgegeben: Punktedatei für den aktuellen Roboter nicht gefunden. Um dies zu korrigieren, fügen Sie die Punktdatei dem Robter im Projekteditor hinzu, oder führen Sie SavePoints zunächst vom Online-Fenster aus aus, nachdem Sie den aktuellen Robter eingestellt haben.

Verwandte Befehle

Dir, Robot, SavePoints

Beispiel einer LoadPoints Anweisung

```
Function main

  ' Lädt gemeinsame Punkte für den aktuellen Roboter.
  LoadPoints "R1Common.pnt"

  ' Hängt Punkte für Modell 1 an
  LoadPoints "R1Model1.pnt", Merge

  Robot 2
  ' Lädt Punktdateien für den Roboter 2
  LoadPoints "R2Model1.pnt"

Fend
```

Local Anweisung

> S

Definiert lokale Koordinatensysteme und zeigt sie an.

Syntax

(1) Local *localNumber*, (*pLocal1 : pBase1*), *(pLocal2 : pBase2)* [, { **L** | **R** }] [, **BaseU**]

(2) Local *localNumber, X, Y, Z, U*

Parameter

localNumber	Die Nummer des lokalen Koordinatensystems. Es kann eine Gesamtzahl von 15 lokalen Koordinatensystemen definiert werden (mit Integer Werten von 1 bis 15).
pLocal1, pLocal2	Punktvariablen mit Punktdaten im lokalen Koordinatensystem.
pBase1, pBase2	Punktvariablen mit Punktdaten im Basiskoordinatensystem.
L | **R**	Optional. Richtet den lokalen Ursprung an dem linken (ersten) oder rechten (zweiten) Basispunkt aus.
BaseU	Optional. Wird BaseU angegeben, befinden sich die U-Achsen-Koordinaten im Basiskoordinatensystem. Wird BaseU weggelassen, befinden sich die U-Achsen-Koordinaten im lokalen Koordinatensystem.
X	Ursprung der X-Achse im lokalen Koordinatensystem.
Y	Ursprung der Y-Achse im lokalen Koordinatensystem.
Z	Ursprung der Z-Achse im lokalen Koordinatensystem.
U	Rotationswinkel des lokalen Koordinatensystems.

Beschreibung

(1) Local definiert ein lokales Koordinatensystem durch die Spezifizierung zweier Punkte, *pLocal1* und *pLocal2*, die darin enthalten sind, und die mit zwei Punkten *pBase1* und *pBase2* übereinstimmen, welche im Basiskoordinatensystem vorhanden sind.

Beispiel:

```
LOCAL 1, (P1:P11), (P2:P12)
```

P1 und P2 sind Punkte im lokalen Koordinatensystem. P1 und P2 sind Punkte im Basiskoordinatensystem.

Wenn die Entfernung zwischen den zwei definierten Punkten im lokalen Koordinatensystem nicht mit der Entfernung zwischen den zwei definierten Punkten im Basiskoordinatensystem übereinstimmt, wird die XY-Ebene des lokalen Koordinatensystem in *der* Position definiert, in welcher der Mittelpunkt zwischen den zwei definierten Punkten im lokalen Koordinatensystem mit dem zwischen den zwei definierten Punkten im Basiskoordinatensystem übereinstimmt.

Gleichermaßen wird die Z-Achse des lokalen Koordinatensystems in *der* Position definiert, in welcher die Mittelpunkte miteinander übereinstimmen.

(2) Definiert ein lokales Koordinatensystem durch Definition von Ursprung und Rotationswinkel im Verhältnis zum Basiskoordinatensystem.

Verwendung der Parameter L und R

Während Local, wie oben beschrieben, im Wesentlichen Mittelpunkte für die Positionierung der Achsen Ihres lokalen Koordinatensystems verwendet, können Sie optional das linke oder rechte Local auch über die Parameter **L** und **R** definieren.

Left Local (Linkes Local)
Left local definiert ein lokales Koordinatensystem durch Definition des Punktes pLocal1 entsprechend Punkt pBase1 im Basiskoordinatensystem (die Z-Achsen-Richtung ist eingeschlossen).

Right Local (Rechtes Local)
Right local definiert ein lokales Koordinatensystem durch Definition des Punktes pLocal2 entsprechend Punkt pBase2 im Basiskoordinatensystem. (Die Z-Achsen-Richtung ist eingeschlossen).

Verwendung des BaseU-Parameters

Wenn der Parameter **BaseU** weggelassen wird, wird die U-Achse des lokalen Koordinatensystems automatisch gemäß der X- und Y-Koordinatenwerte der definierten vier Punkte korrigiert. Die zwei Punkte im Basiskoordinatensystem können daher zunächst U-Koordinaten-Werte haben.

Es kann der Wunsch bestehen, die U-Achse des lokalen Koordinatensystems auf der Grundlage der U-Koordinaten-Werte der zwei Punkte im Basiskoordinatensystem selbst zu korrigieren, anstatt sie automatisch korrigieren zu lassen (z.B. Korrektur der Rotationsachse durch Teach-In). Um dies zu erreichen, geben Sie bitte den **BaseU**-Parameter an.

Hinweise

Unter folgenden Bedingungen gehen lokale Definitionen verloren:

Die Definitionen der lokalen Koordinatensysteme 1 bis 15 gehen verloren, wenn der Strom abgeschaltet wird, oder wenn die Befehle Base oder Verinit ausgeführt werden.

Verwandte Befehle

Base

Local Beispiel

Hier sind einige Beispiele vom Online-Fenster:

```
> p1 = 0, 0, 0, 0/1
> p2 = 100, 0, 0, 0/1
> p11 = 150, 150, 0, 0
> p12 = 300, 150, 0, 0
> local 1, (P1:P11), (P2:P12), L

> p21 = 50, 0, 0, 0/1
> go p21
```

Local Funktion

F

Syntax

Local(*localNumber*)

Parameter

localNumber Nummer des lokalen Koordinatensystems (ganze Zahl von 0 bis 15) repräsentiert durch einen Ausdruck oder einen numerischen Wert.

Rückgabewerte

Spezifizierte Daten des lokalen Koordinatensystems als Punktdaten.

Verwandte Befehle

Local Anweisung

Beispiel einer Local Funktion

```
P1 = XYLim(1)
```

Lof Funktion

F

Überprüft, ob der angegebene RS-232 oder TCP/IP Port Datensätze hat.

Syntax

Lof(*portNumber*)

Parameter

portNumber Nummer der Kommunikationsschnittstelle.

Rückgabewerte

Die Anzahl der Datensätze im Puffer. Wenn sich keine Daten im Puffer befinden, beträgt der Rückgabewert von **Lof** Null (0).

Beschreibung

Lof überprüft, ob die angegebenen Schnittstelle Daten empfangen hat, oder nicht. Die empfangenen Daten werden unabhängig vom Input#-Befehl im Puffer gespeichert.

Verwandte Befehle

ChkCom, ChkNet, Input#

Beispiel einer Lof Funktion

Dieses Online-Fenster-Beispiel druckt die Anzahl der durch die Kommunikationsschnittstelle Nr.1 empfangenen Datensätze aus.

```
>print lof(1)
 5
>
```

LogIn Anweisung

S

Loggt als anderer Anwender in EPSON RC+ ein.

Syntax

LogIn *logID, password*

Parameter

logID	Zeichenkettenausdruck, der die Anwender-Login-ID enthält.
password	Zeichenkettenausdruck, der das Anwender-Passwort enthält.

Beschreibung

Sie können EPSON RC+ Security in Ihrer Anwendung verwenden. Beispielsweise können Sie ein Menü anzeigen, das es verschiedenen Anwendern gestattet, sich in das System einzuloggen. Jeder Anwender-Typ kann seine eigenen Security-Rechte haben. Für weitere Details bezüglich der Sicherheitsvorkehrungen lesen Sie bitte das EPSON RC+ Benutzerhandbuch.

Wenn Sie Programme in der Development-Umgebung ausführen, wird nach Beendigung des Programms automatisch der zuletzt verwendete Anwendername angegeben.

Wenn Sie das Operator-Fenster mit den /OPR oder /OPRAS Befehlszeilenoptionen ausführen, wird die Anwendung als Gast-Anwender eingeloggt, es sei denn, Auto LogIn ist aktiviert. Ist dies der Fall, so ist die Anwedung als aktueller Windows-Anwender eingeloggt, wenn ein solcher Anwender im EPSON RC+ System konfiguriert wurde.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die “Security” Option („Sicherheitsoption“) installiert ist.

Verwandte Befehle

GetCurrentUser$ Funktion

Beispiel einer LogIn Anweisung

```
LogIn "operator", "oprpass"
```

Long Anweisung

S

Deklariert Long Integer Variablen. (4 Byte Integer).

Syntax

Long *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]**...**

Parameter

varName	Variablenname, den der Anwender als **Long** deklarieren will.
subscripts	Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt **(***dim1* [, *dim2* [, *dim3*]]**)** *dim1*, *dim2*, *dim3* können eine ganze Zahl zwischen 0-32767 sein.

Beschreibung

Long wird verwendet, um Variablen als **Long** zu deklarieren. **Long** -Variablen können ganze Zahlen im Wertebereich zwischen -2.147.483.648 und 2.147.483.647 enthalten. Alle Variablen sollten ganz oben in einer Funktion deklariert sein.

Verwandte Befehle

Boolean, Byte, Double, Global, Integer, Real, String

Beispiel einer Long Anweisung

Das folgende Beispiel zeigt ein einfaches Programm, das unter Verwendung von **Long** einige Variablen als Long deklariert.

```
Function longtest
    Long A(10)         'Eindimensionale Feldvariable aus Longs
    Long B(10, 10)     'Zweidimensionale Feldvariable aus Longs
    Long C(10, 10, 10) 'Dreidimensionale Feldvariable aus Longs
    Long var1, arrayVar(10)
    Long i
    Print "Please enter a Long Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For I = 1 To 5
        Print "Please enter a Long Number"
        Input arrayVar(i)
        Print "Value Entered was ", arrayVar(i)
    Next I
Fend
```

LPrint Anweisung

> S

Der **LPrint**-Befehl sendet ASCII-Text an den PC-Drucker.

Syntax

LPrint [*expression,* [, *expression...*]

Parameter

expression Optional. Ein numerischer- oder ein Zeichenkettenausdruck.

Beschreibung

Verwenden Sie **Lprint,** um eine Daten-Hardcopy auf dem Vorgabe-PC-Drucker zu drucken. Nach der Ausführung von einer oder mehreren **LPrint–**Anweisungen, müssen Sie EPrint ausführen, um die Daten an den Drucker zu senden.

Ans Ende jeder Zeile wird automatisch die Endung CRLF (carriage return and line feed) angehängt.

Verwandte Befehle

EPrint, Print

Beispiel einer LPrint Anweisung

```
Function dataPrint
    Integer i

    For i = 1 to 10
        LPrint g_testDat$(i)
    Next i
    EPrint   ' Schickt die Daten an den Drucker des PCs
Fend
```

LShift Funktion

F

Verschiebt numerische Daten um eine vom Anwender definierte Bitanzahl nach links.

Syntax

LShift*(number, shiftBits)*

Parameter

number	Der Integer-Ausdruck, der verschoben werden soll.
shiftBits	Bitanzahl, um die *number* nach links verschoben werden.

Rückgabewerte

Gibt ein numerisches Ergebnis aus, das dem *number*-Wert entspricht, nachdem die Bits um die in *shiftBits* definierte Anzahl von Bits nach links verschoben wurden.

Beschreibung

LShift verschiebt die spezifizierten numerischen Daten (*number*) um die festgelegte Bitanzahl (*shiftBits*) nach links (an eine höherwertige Stelle). Die nachrückenden Bits niedrigeren Wertes werden 0 gesetzt.

Die einfachste Erklärung für LShift ist, dass es einfach das Ergebnis von $number * 2^{shiftBits}$ ausgibt. (*Number* wird multipliziert mit 2 hoch *shiftBit*)

Hinweis

Typ numerischer Daten:

Die numerischen Daten (*num*) können aus jeglichem gültigen numerischen Datentyp bestehen. **LShift** arbeitet mit den Datentypen: Byte, Integer und Real.

Verwandte Befehle

And, Not, Or, RShift, Xor

Beispiel einer LShift Funktion

```
Function lshiftst
    Integer i
    Integer num, snum
    num = 1
    For i = 1 to 10
         Print "i =", i
         snum = LShift(num, i)
         Print "The shifted num is ", snum
    Next i
Fend
```

Einige weitere Beispiele des LShift-Befehls vom Online-Fenster.

```
> Print LShift(2,2)
8
> Print LShift(5,1)
10
> Print LShift(3,2)
12
>
```

LTrim$ Funktion

F

Gibt eine Zeichenkette aus, die mit der definierten Zeichenkette identisch ist, ohne vorangestellte Leerzeichen.

Syntax

LTrim$ (*string*)

Parameter

string Zeichenkettenausdruck.

Rückgabewerte

Spezifizierte Zeichenkette, bei der die vorangehenden Leerzeichen entfernt wurden.

Verwandte Befehle

RTrim$

Beispiel einer LTrim$ Funktion

```
str$ = "  data  "
str$ = LTrim$(str$)  ' str$ = "data  "
```

MCal Anweisung

INC > S

Führt die Maschinenkalibrierung für Roboter mit inkrementellen Encodern durch.

Syntax

MCal

Beschreibung

Es ist notwendig, Roboter mit inkrementellen Encodern zu kalibrieren. Die Kalibrierung muss nach Einschalten der Versorgungsspannung ausgeführt werden. Wenn Sie versuchen, einen Bewegungsbefehl oder einen beliebigen anderen Befehl auszuführen, für den die aktuellen Positionsdaten benötigt werden, ohne zuerst die Maschinenkalibrierung durchzuführen, tritt ein Fehler auf.

Die Maschinenkalibrierung wird in der Reihenfolge der Achsbewegungen durchgeführt, welche mit dem MCordr-Befehl definiert ist. Der Vorgabewert von MCordr zum Zeitpunkt der Lieferung variiert von Modell zu Modell. Lesen Sie daher bitte das entsprechende Roboterhandbuch bezüglich weiterer Details.

Normalerweise wird MCal ausgeführt, wenn sich der Roboter im Zentrum des Bewegungs-Arbeitsbereiches befindet. Bei SCARA-Robotern, die mehrfache Kalibrierungspunkte haben, ist die kombinierte Bewegung von erster und zweiter Achse während Mcal auf +/-15-16 Grad im oder gegen den Uhrzeigersinn eingeschränkt. Dies hilft, die zur Kalibrierung des Roboters notwendige Zeit zu reduzieren. Wenn Mcal in Nähe der Arbeitsbereichsgrenze ausgeführt wird, kann es sein, dass der Arm versucht, sich aus dem Bereich heraus zu bewegen. In diesem Fall kann ein Fehler ausgegeben werden, oder der Arm fährt gegen den mechanischen Anschlag. Um Mcal auszuführen nachdem dies geschehen ist, schalten Sie einfach die Motoren aus und bewegen Sie den Arm von Hand aus der Grenze des Arbeitsbereiches.

Die tatsächliche Bewegungsentfernung bei der Maschinenkalibrierung ist von Modell zu Modell unterschiedlich. Bitte lesen Sie das entsprechende Roboter-Handbuch für weitere Details.

Potentielle Fehler

Für SCARA Roboter mit inkrementellen Encodern:

Bei SCARA Robotern mit inkrementellen Encodern sollte die vierte Achse nicht um mehr als 180 Grad rotiert werden, wenn die Steuerung ausgeschaltet ist. Wenn Sie nach Rotation der Achse die Versorgungsspannung einschalten und Mcal ausführen, tritt Fehler 237 auf. In diesem Fall sollten Sie die Versorgungsspannung ausschalten und die vierte Achse ungefähr in die vorangegangene Position bewegen, dann den Strom einschalten und erneut versuchen, **Mcal** auszuführen.

Der Versuch, einen Bewegungsbefehl auszuführen, ohne zuvor Mcal ausgeführt zu haben:

Wenn Sie versuchen, einen Bewegungsbefehl oder einen beliebigen anderen Befehl auszuführen, für den die aktuellen Positionsdaten (z.B. der Plist* Befehl) benötigt werden, ohne zuerst die Maschinenkalibrierung durchzuführen, tritt ein Fehler auf.

Roboter mit absoluten Encodern:

Wenn Sie versuchen, Mcal für Roboter mit absoluten Encodern auszuführen, tritt Fehler Nr. 123 auf, was bedeutet, das dieser Befehl für diesen Robotertyp nicht unterstützt wird. Roboter mit absoluten Encodern benötigen die Mcal-Anweisung nicht.

Hinweis zur Roboterinstallation

Z Achsen-Raum wird für den Homing-Prozess benötigt:

Wenn die Z-Achse in ihre Ausgangsposition zurückkehrt, fährt sie zunächst aufwärts, dann abwärts und bewegt sich dann in die Home-Position.. Dies bedeutet, dass es sehr wichtig ist, den Roboter korrekt zu installieren, so dass genug Platz vorhanden ist, damit der Arm in Z-Position in seine Home-Position gelangen kann. Es wird empfohlen, über dem oberen Limit 6 mm Raum zu lassen. (Installieren Sie keine Werkzeuge oder Einbauten im Raum von 6 mm über dem Roboter, damit für Z-Achsen-Rückkehr in die Home-Position genügend Platz vorhanden ist.)

Verwandte Befehle

Hofs, HTest, Home, Hordr, Mcorg, MCOrdr, MCofs

Mcal Beispiel

Das folgende Beispiel wird vom Online-Fenster aus ausgeführt:

```
> Motor On
> Mcal
>
```

MCalComplete Funktion

F

Gibt den Status von MCal aus.

Syntax

MCalComplete

Rückgabewerte

WAHR, wenn MCal abgeschlossen wurde, sonst FALSCH.

Verwandte Befehle

MCal

Beispiel von MCalComplete

```
If Not MCalComplete Then
    MCal
EndIf
```

MCofs Anweisung

INC >

Spezifiziert die Kalibrierungs-Offsetparameter für die Maschinenkalibrierung und zeigt sie an. Nur benötigt für Roboter mit inkrementellen Encodern.

Syntax

MCofs *sensorLogic, j1Pulses, j2Pulses, j3Pulses, j4Pulses, j1WDiff, j2WDiff*

Parameter

sensorLogic	Integer-Ausdruck, welcher die Sensorlogik repräsentiert.
j1Pulses	Die Pulse-Werte zwischen dem Einschalten des Sensors und Erkennen der Z-Phase am definierten Punkt der ersten Achse.
j2Pulses	Die Pulse-Werte zwischen dem Einschalten des Sensors und Erkennen der Z-Phase am definierten Punkt der zweiten Achse.
j3Pulses	Die Pulse-Werte zwischen dem Einschalten des Sensors und Erkennen der Z-Phase am definierten Punkt der dritten Achse.
j4Pulses	Die Pulse-Werte zwischen dem Einschalten des Sensors und Erkennen der Z-Phase am definierten Punkt der vierten Achse.
j1WDiff	Der Unterschied zwischen dem logischen Wert und der Breite der Sensorkante an der definierten Position der ersten Achse.
j2WDiff	Der Unterschied zwischen dem logischen Wert und der Breite der Sensorkante an der definierten Position der zweiten Achse.

Rückgabewerte

Zeigt die aktuellen MCofs-Werte an, wenn die Parameter weggelassen werden.

Beschreibung

Definiert die Parameter für die Maschinenkalibrierung. Dieser Parameter ist notwendig, um die Maschinenkalibrierung mit Mcal durchzuführen. Wenn Sie alle Spezifikationswerte definieren, werden diese als Parameter für die Maschinenkalibrierung eingesetzt.

Die Parameter für die Kalibrierung werden automatisch berechnet, wenn der Roboterarm mit Mcorg bewegt wird. Es wird empfohlen, Mcorg zu verwenden, um Mcofs-Werte zu erzeugen. Wenn diese Werte jedoch im Voraus bekannt sind, können Sie diese Werte über den MCofs-Befehl eingeben.

Hinweise

WICHTIG: MCofs sollte nur für Wartungszwecke verwendet werden:

Wenn die Maschinenkalibrierung mit Mcal ausgeführt wird, so wird die aktuelle Position auf diesem Parameter basierend erkannt. Diese Daten müssen daher angegeben werden. Wenn die falschen Daten definiert werden, sind die Roboterkoordinaten nicht korrekt und die falschen Daten lösen eine unerlaubte Bewegung aus.

Abschalten der Versorgungsspannung und Ausführen des Verinit-Befehls

Die MCofs-Werte werden sowohl bei Abschalten der Versorgungsspannung, als auch nach Ausführung des Verinit-Befehls beibehalten.

Roboter mit absoluten Encodern:

Wenn Sie versuchen, MCOFS für Roboter mit absoluten Encodern auszuführen, so tritt Fehler Nr. 409 auf, was bedeutet, das dieser Befehl für diesen Robotertyp nicht unterstützt wird. Roboter mit absoluten Encodern benötigen die MCOFS-Anweisung nicht.

Verwandte Befehle

Mcal, Mcorg, Ver

Beispiel einer MCofs Anweisung

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt:

```
> MCofs
04
1364  50
4506  6304
-479  -469

> MCofs &H04, 1364, 50, 4506, 6304, -479, -469
```

MCofs Funktion

F

Syntax

MCofs(*paramNumber*)

Parameter

paramNumber Definiert die MCOFS-Einstellungsnummern (ganze Zahl von 0 bis 7), dargestellt durch einen Ausdruck oder einen numerischen Wert.

Rückgabewerte

Gibt den für die Maschinenkalibrierung definierten Parameter aus.

Verwandte Befehle

Mcal, Mcorg, Ver

Beispiel einer MCofs Funktion

```
Integer a

a = MCofs(1)
```

MCordr Anweisung

INC > S

Definiert die Bewegungs-Achsreihenfolge für die Maschinenkalibrierung Mcal und zeigt sie an. Wird nur für Roboter mit inkrementellen Encodern benötigt.

Syntax

MCordr [*Step1, Step2, Step3, Step4*]

Parameter

Step1 Bitmuster, aus dem hervorgeht, welche Achsen während des 1. Schrittes des MCal-Prozesses kalibriert werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 1. Schrittes kalibriert werden. (bzgl. der Definition von Bitmustern: siehe unten)

Step2 Bitmuster, aus dem hervorgeht, welche Achsen während des 2. Schrittes des MCal-Prozesses kalibriert werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 2. Schrittes kalibriert werden. (bzgl. der Definition von Bitmustern: siehe unten)

Step3 Bitmuster, aus dem hervorgeht, welche Achsen während des 3. Schrittes des MCal-Prozesses kalibriert werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 3. Schrittes kalibriert werden. (bzgl. der Definition von Bitmustern: siehe unten)

Step4 Bitmuster, aus dem hervorgeht, welche Achsen während des 4. Schrittes des MCal-Prozesses kalibriert werden sollen. Eine beliebige Anzahl von Achsen von 0 bis zu allen 4 kann während des 4. Schrittes kalibriert werden. (bzgl. der Definition von Bitmustern: siehe unten)

Rückgabewerte

Zeigt die aktuelle Maschinenkalibrierungs-Reihenfolge an, wenn die Parameter weggelassen werden.

Beschreibung

Nachdem die Versorgungsspannung am System eingeschaltet wurde, oder nach Ausführung des Verinit-Befehls, muss der Mcal-Befehl ausgeführt werden, bevor eine Operation mit dem Roboterarm ausgeführt wird. Wenn der Mcal-Befehl ausgeführt wird, fährt jede der vier Roboterachsen in ihre entsprechende Kalibrierungsposition.

Definiert die Bewegungsreihenfolge für den Mcal-Befehl. (D.h. die Anweisung definiert, welche Achse zuerst in ihre Home-Position zurückgeführt werden soll, und welche als 2., 3., usw. kalibriert wird)

Der Zweck des **MCordr-**Befehls ist es, dem Anwender zu ermöglichen, die Reihenfolge zu ändern, in der die Achsen in ihre Home-Position zurückgeführt werden sollen. Die Homing-Reihenfolge ist in vier unterschiedliche Schritte unterteilt. Der Anwender verwendet **MCordr** dann, um die spezifischen Achsen zu definieren, die sich während eines jeden Schrittes (mit dem Mcal-Befehl) in die Kalibrierungsposition bewegen. Es ist wichtig, zu verstehen, dass mehr als eine Achse definiert werden kann, die sich während eines einzigen Schrittes in die Kalibrierungsposition bewegen soll. Dies bedeutet, dass es möglich ist, alle vier Achsen zur selben Zeit in ihre Kalibrierungsposition zurückzuführen. Es wird jedoch empfohlen, im Normalfall zunächst die Z-Achse zu ihrer Kalibrierungsposition zu bewegen (in Step/Schritt 1) und die anderen Achsen in den darauf folgenden Steps/Schritten folgen zu lassen. (Siehe unten stehenden Hinweis)

Der **MCordr** Befehl setzt voraus, dass ein Bitmuster für jeden der vier Schritte definiert wurde. Da es vier Achsen gibt, wird jeder Achse ein spezifisches Bit zugeordnet. Wenn das Bit eingeschaltet ist (1) (für einen spezifischen Schritt), wird die entsprechende Achse kalibrieren. Wenn das Bit ausgeschaltet ist (0), wird die entsprechende Achse während dieses Schrittes nicht kalibrieren. Die Achsen-Bitmuster werden zugeordnet wie folgt:

Achse:	**1. Achse (T1)**	**2. Achse (T2)**	**3. Achse (Z)**	**4. Achse (U)**
Bitnummer:	Bit 0	Bit 1	Bit 2	Bit 3
Binärcode:	&B0001	&B0010	&B0100	&B1000

Hinweise

Der Unterschied zwischen MCordr und Hordr:

Obwohl die Befehle Hordr und **MCordr** auf den ersten Blick recht ähnlich aussehen, gibt es zwischen ihnen einen Hauptunterschied, den man verstehen muss. **MCordr** wird verwendet, um die Achsenreihenfolge für die Roboterkalibrierung zu definieren (wird mit Mcal verwendet) während **Hordr** verwendet wird, um die Achsenreihenfolge für das Homing zu definieren (wird mit dem Home Befehl verwendet).

Vorgegebene Homing-Reihenfolge (werksseitige Einstellung):

Werksseitig ist die Kalibrierungsreihenfolge dahingehend eingestellt, dass die Z-Achse im 1. Schritt in ihre Home-Position zurückgeführt wird. Im 2. Schritt werden dann die Achsen 1, 2, und U gleichzeitig in ihre Home-Position zurückgeführt. (Die Schritte 3 und 4 werden in der Vorgabekonfiguration nicht verwendet). Die vorgegebenen **MCordr**-Werte lauten wie folgt:

MCordr &B0100, &B1011, 0, 0

Der Verinit Befehl (Zurücksetzen der vorgegebenen Hordr-Werte)

Die **MCordr -**Werte werden durch Ausführen des Verinit-Befehls auf die o.a. Werte zurückgesetzt.

Die Z-Achse sollte im Normalfall zuerst kalibriert werden.

Der Grund, warum die Z-Achse als Erste (und ausschließlich) bewegt werden soll liegt darin, dass das Roboterwerkzeug über die Arbeitsfläche bewegt werden soll, bevor mit einer Horizontalbewegung begonnen wird. Dies trägt dazu bei, zu verhindern, mit dem Roboterwerkzeug während des Homing-Prozesses im Arbeitsbereich gegen etwas zu stoßen.

Die MCordr-Werte werden beibehalten

Die Werte der **MCordr**-Tabelle werden dauerhaft gespeichert und nicht geändert, bis sie entweder vom Anwender geändert werden, oder ein Verinit-Befehl ausgegeben wird.

Verwandte Befehle

Mcal, MCofs

Beispiel einer MCordr Anweisung

Im Folgenden sehen Sie einige Online-Fenster-Beispiele:

Dieses Beispiel instruiert den Roboterarm, die Kalibrierungsreihenfolge wie folgt zu definieren: Z-Achse im ersten Schritt, 1. Achse im zweiten Schritt, 2. Achse im dritten Schritt, und dann die U-Achse im vierten Schritt. Die Reihenfolge ist in Bitmustern spezifiziert.

```
> MCordr  &B0100, &B0001, &B0010, &B1000
```

Dieses Beispiel instruiert den Roboterarm, die Z-Achse als erste zu kalibrieren (in Step/Schritt 1), danach die Achsen 1, 2 und U gleichzeitig (&B1011, oder 11 als Dezimalwert) in Step/Schritt 2. Die Maschinenkalibrierungs-Reihenfolge ist in Dezimalwerten spezifiziert.

```
> MCordr  4, 11, 0, 0
```

Dieses Beispiel zeigt die aktuelle Maschinenkalibrierungs-Reihenfolge in Dezimalzahlen.

```
>mcordr
4, 11, 0, 0
>
```

MCordr Funktion

F

Syntax

MCordr(*paramNumber*)

Parameter

paramNumber Definiert die Referenz-Einstellungsnummern (Integers von 1 bis 4) repräsentiert durch einen Ausdruck oder einen numerischen Wert.

Rückgabewerte

Gibt Binärwerte aus (Integers), die die Achsen der definierten Einstellungsnummer repräsentieren, welche die Maschinenkalibrierung ausführen sollen.

Beschreibung

Gibt die Achsen-Bewegungsreihenfolge aus, um mit Mcal die Maschinenkalibrierung durchzuführen.

Verwandte Befehle

Mcal, MCofs

Beispiel einer MCordr Funktion

Dieses Beispiel benutzt die **MCordr**-Funktion in einem Programm:

```
Integer a
a = MCordr(1)
```

Mcorg Anweisung

INC >

Berechnet die korrekten Kalibrierungsparameter für die Maschinenkalibrierung (Mcal).

Syntax

Mcorg *jointNumber* [,*jointNumber* [,*jointNumber* [,*jointNumber*]]]

Parameter

jointNumber Integer-Ausdruck zwischen 1 und 4, der die Achse repräsentiert, für die **Mcorg** ausgeführt werden soll. Es muss mindestens eine Achsnummer definiert werden, damit der **Mcorg-**Befehl korrekt und fehlerfrei funktioniert. Mehrere Achsen können **Mcorg** zur selben Zeit durchführen lassen. (Wenn die Roboter eine Kugelumlaufspindel haben, wie es zum Beispiel bei SCARA Robotern der Fall ist, und die Notwendigkeit besteht, für die dritte und vierte Achse einen Kalibirerungsparameter zu berechnen, spezifizieren Sie bitte beide Achsen zusammen. Wenn nur eine Achse spezifiziert wird, tritt ein Fehler auf).

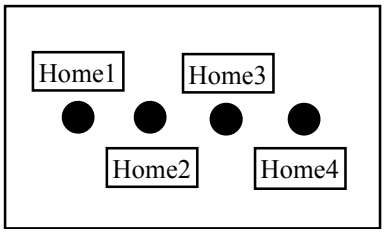
Beschreibung

Berechnet die korrekten Kalibrierungsparameter für die Maschinenkalibrierung mit Mcal. Die Schritte, die notwendig sind, um Mcorg mit SCARA Robotern richtig zu nutzen, werden unten gezeigt:

1) Stellen Sie sicher, dass die Motoren ausgeschaltet sind. [Führen Sie den “Motor Off” (Motor Aus) –Befehl aus]

2) Bewegen Sie die dritte und vierte Roboterachse innerhalb des Bewegungsbereiches.

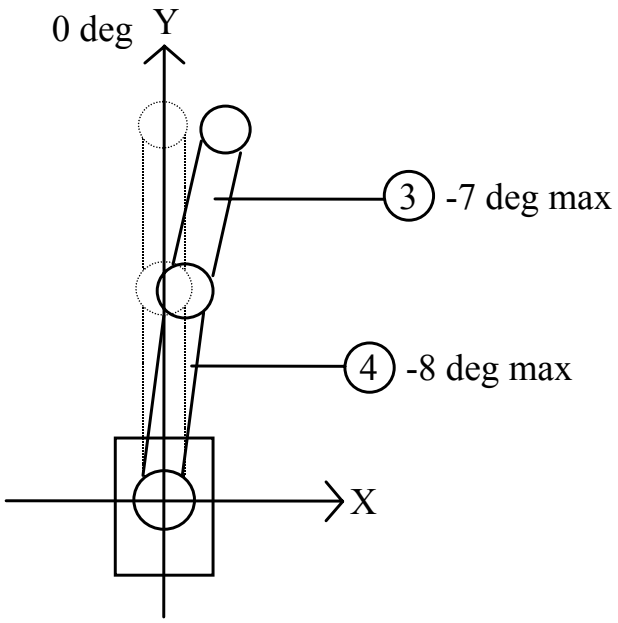
3) Bewegen Sie den Arm in die korrekte **Mcorg** Position indem Sie die LEDs auf dem Sensormonitor verwenden, welcher sich auf der Rückseite der Roboterbasis befindet. Die LEDs werden unten angezeigt: Die Achsnummern stehen neben jeder LED, um anzuzeigen, welche LED welcher Kalibrierungs-LED entspricht. Wenn Sie die erste oder zweite Achse bewegen, schaltet sich die der Achse entsprechende LED in Übereinstimmung mit der Armbewegung ein oder aus.



Strecken Sie den Arm aus, wie in der gepunkteten Linie unten gezeichnet, so dass er parallel zur +Y-Achse der Roboterkoordinaten steht. (D.h. direkt von der Basis wegzeigend). In dieser Position schalten sich die LEDs HOME1 und HOME2 ein.

4) Während Sie die LED HOME2 beobachten, bewegen Sie das zweite Armgelenk von Hand im Uhrzeigersinn. Während Sie das zweite Armgelenk bewegen, halten Sie das erste Armgelenk fest, so dass es sich nicht bewegt. Stoppen Sie die Bewegung des zweiten Armgelenks ungefähr in der Mitte des Bereichs, in dem die HOME2 LED sich zum ersten Mal ausschaltet.

5) Als nächstes, während Sie die LED HOME1 beobachten, bewegen Sie das erste Armgelenk von Hand im Uhrzeigersinn. Stoppen Sie die Bewegung des ersten Armgelenks ungefähr in der Mitte des Bereichs, in dem die HOME1 LED sich zum ersten Mal ausschaltet. Der ungefähre Mittelpunkt sollte sich ca. 8 Grad von der Mittellinie der Y-Achse entfernt befinden. Der ungefähre Mittelpunkt des zweiten Armgelenks wird sich ca. 7 Grad von der Mittellinienposition der ersten Gelenksachse entfernt befinden. Sie unten stehende Zeichnung:



6) Schalten Sie mit Hilfe des “Motor On”-Befehls die Motoren ein.

7) Wenn Sie in dieser Position den **Mcorg** –Befehl ausführen, bewegen sich die erste und zweite Achse gegen den Uhrzeigersinn. Die maximale Bewegung sollte +15 für Achse 2 und +18 für Achse 1 betragen. Beseitigen Sie jegliche Hindernisse aus dem Bereich, bevor Sie **Mcorg** ausführen.

8) Führen Sie den Mcal-Befehl aus, um die Maschinenkalibrierung durchzuführen.

9) Führen Sie den Pulse-Befehl aus, um den Arm in die 0-Pulse-Position der ersten und zweiten Achse zu bringen.

```
> Pulse 0, 0, 0, 0
```

10) Nachdem sich der Arm in die Pulse-Position 0, 0, 0, 0 bewegt hat, stellen Sie sicher, dass die Armachsen 1 und 2 gerade sind und sich auf der X-Achse des Roboterkoordinatensystems befinden. Sollte dies nicht der Fall sein, bedeutet dies, dass die Armposition zu dem Zeitpunkt, an dem **Mcorg** ausgeführt wurde, nicht korrekt war. Gehen Sie in diesem Falle zurück zu Punkt 2 und beginnen Sie erneut.

Hinweise

Was passiert, wenn Mcorg nicht in der korrekten Position ausgeführt wird:

Bei Robotern mit mehreren Kalibrierungspunkten, wie z.B. den Robotern der TT8000 Serie, werden die Kalibrierungsparameter nicht korrekt berechnet, wenn **Mcorg** nicht an der richtigen Stelle ausgeführt wird. Wenn die Parameter falsch sind, bewegt sich der Roboter nicht richtig.

Wenn Sie **Mcorg** versehentlich ausführen, führen Sie keine Bewegungsbefehle aus. In dieser Situation sollten Sie **Mcorg** nochmals von der korrekten Position ausführen, oder die bei Lieferung ursprünglich eingestellten MCofs Daten finden, und diese Daten eingeben, um den MCofs –Befehl zu verwenden.

Verwandte Befehle

Mcal, MCofs

Beispiel einer Mcorg Anweisung

Das folgende Beispiel wurde vom Online-Fenster aus ausgeführt:

```
> mcorg 1,2   'Berechnet die Parameter für die Maschinenkalibrierung
              'der ersten und zweiten Achse
```

Mid$ Funktion

F

Gibt eine Zeichenkette der mittleren *count*-Buchstaben der Zeichenkette *string* aus, beginnend mit dem *position* -Buchstaben.

Syntax

Mid$*(string, position, count*)

Parameter

string	Quell-Zeichenkettenausdruck.
position	Die Startposition in der Buchstaben-Zeichenkette für das Kopieren von *count*-Buchstaben.
count	Anzahl der zu kopierenden Zeichen aus *string,* beginnend mit dem durch *position* definierten Zeichen.

Rückgabewerte

Gibt eine Teilkette von Zeichen aus *string* aus.

Beschreibung

Mid$ gibt eine Teilkette von so vielen *count* Zeichen aus, wie mit dem *position* Zeichen in *string* beginnen.

Verwandte Befehle

Asc, Chr$, Left$, Len, Right$, Space$, Str$, Val

Beispiel einer Mid$ Funktion

Das Beispiel unten zeigt ein Programm, welches die zwei mittleren Buchstaben aus der Zeichenkette "ABCDEFGHIJ" zieht.

```
Function midtest
    String basestr$, middle$
    basestr$ = "ABCDEFGHIJ"
    middle$ = Mid$(basestr$, (Len(basestr$) / 2), 2)
    Print "The middle 2 characters are: ", middle$
Fend
```

Einige weitere Beispiele des **Mid$**-Befehls vom Online-Fenster.

```
> Print Mid$("ABCDEFG", 4, 2)
 DE

> Print Mid$("ABCDEF", 1, 3)
 ABC
>
```

MkDir

>

Erstellt ein Unterverzeichnis auf dem aktuellen PC-Laufwerk.

Syntax

MkDir *dirName*

Parameter

dirName	Zeichenkettenausdruck, der den Namen des zu erstellenden Verzeichnisses definiert. Sie können auch einen Pfad mit einbeziehen.

Beschreibung

Erstellt ein Unterverzeichnis im spezifizierten Pfad.

Es kann immer nur ein Unterverzeichnis gleichzeitig erstellt werden.

Wenn dieser Vorgang vom Online-Fenster aus getätigt wird, können Sie die Anführungsstriche auslassen.

Verwandte Befehle

ChDir, ChDrive, Dir, Rendir

MkDir Beispiel

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt:

```
> mkdir \USR
> mkdir \USR\PNT
```

Mod Operator

> S

Gibt den Restbetrag aus, den man erhält, indem man einen numerischen Ausdruck durch einen anderen dividiert.

Syntax

number **Mod** *divisor*

Parameter

number Die Zahl, die geteilt wird. (Der Dividend).

divisor Die Zahl, durch welche *number* geteilt wird

Rückgabewerte

Gibt den verbleibenden Rest, nachdem *number* durch *divisor* geteilt wurde aus.

Beschreibung

Mod wird verwendet, um den verbleibenden Restbetrag zu erhalten, nachdem man 2 Zahlen dividiert hat. Bei dem Rest handelt es sich um eine ganze Zahl. Eine geschickte Verwendung des **Mod**-Befehls ist es, festzustellen, ob eine Zahl gerade oder ungerade ist. Die Methode, mit der der **Mod-**Befehl arbeitet, ist folgende: *number* wird durch *divisor* geteilt. Der nach dieser Division verbleibende Rest ist dann der Rückgabewert für den **Mod**-Befehl.

Verwandte Befehle

Abs, Atan, Atan2, Cos, Int, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Beispiel für den Mod Operator

Das Beispiel unten zeigt an, ob eine Zahl (var1) gerade oder ungerade ist. Wenn die Zahl GERADE ist, gibt das Ergebnis des Mod-Befehls eine 0 aus. Ist die Zahl UNGERADE, gibt das Ergebnis des Mod-Befehls eine 1 aus.

```
Function modtest
....Integer var1, result

....Print "Enter an integer number:"
....Input var1
....result = var1 Mod 2
....If result = 0 Then
........Print "Result = ", result
........Print "The number is EVEN"
....Else
........Print "Result = ", result
........Print "The number is ODD"
....EndIf
Fend
```

Einige weitere Beispiele des Mod-Befehls vom Online-Fenster.

```
> Print  36 Mod 6
> 0

> Print  25 Mod 10
> 5
>
```

Motor Anweisung

> S

Schaltet den Motorenstrom für alle Achsen des aktuellen Roboters Ein oder Aus.

Syntax

Motor ON | OFF

Parameter

ON | OFF Das Schlüsselwort **ON** wird verwendet, um den Motorenstrom einzuschalten. Das Schlüsselwort **OFF** wird verwendet, um den Motorenstrom auszuschalten.

Beschreibung

Der Befehl **Motor** On wird verwendet, um den Motorenstrom einzuschalten und die Bremsen für alle Achsen freizuschalten. **Motor** Off wird verwendet, um den Motorenstrom auszuschalten und die Bremsen anzuziehen.

In einem Multi-Roboter-System sollten Sie den Robot-Befehl verwenden, um den aktuellen Roboter auszuwählen, bevor Sie den **Motor** Befehl ausführen.

Um den Roboter bewegen zu können, muss der Motorenstrom eingeschaltet sein. Bei Robotern mit inkrementellen Encodern muss nach dem Systemstart auch noch der Mcal-Befehl ausgeführt werden.

Führen Sie Reset nach einem Not-Halt aus, oder nachdem ein Fehler aufgetreten ist, welcher ein Reset mit dem Reset-Befehl notwendig macht, und führen Sie dann **Motor** On aus.

Motor On stellt die folgenden Werte automatisch ein:

Power	Niedrig
Fine	Vorgabewerte
Speed	Vorgabewerte
SpeedS	Vorgabewerte
Accel	Vorgabewerte
AccelS	Vorgabewerte

Verwandte Befehle

Power, Reset, Robot, SFree, SLock

Beispiel der Motor Anweisung

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt:

```
> Motor On

> Motor Off
```

Motor Funktion

F

Gibt den Status der Motore aus.

Syntax

Motor

Rückgabewerte

0 = Motore aus, 1 = Motore eingeschaltet.

Verwandte Befehle

Motor Anweisung

Beispiel der Motor Funktion

```
If Motor = Off Then
    Motor On
EndIf
```

Move Anweisung

> S

Bewegt den Arm von seiner aktuellen Position mittels linearer Interpolation an den definierten Punkt (d.h. Bewegung in einer geraden Linie bei gleich bleibender Werkzeugmittelpunkt-Geschwindigkeit).

Syntax

Move *destination* [**CP**] [*searchExpr*] [*!...!*]

Parameter

destination	Der Zielort einer Bewegung, die einen Punktausdruck verwendet.
CP	Optional. Spezifiziert die Continuous Path (CP / kontinuierlicher Weg) - Bewegung.
searchExpr	Optional. Ein Till oder Find -Ausdruck. **Till** | **Find** **Till Sw**(*expr*) = {**Ein** | **Aus**} **Find Sw**(*expr*) = {**Ein** | **Aus**}
!...!	Optional. Parallelbearbeitungsanweisungen können hinzugefügt werden, um E/A und andere Befehle während der Bewegung auszuführen.

Beschreibung

Move bewegt den Arm in einer geraden Linie von seiner aktuellen Position zu seiner *destination*. **Move** koordiniert alle 4 Achsen dahingehend, dass sie zur selben Zeit starten und stoppen. Die *destination*-Koordinaten müssen vor der Ausführung des Move-Befehls eingeteacht worden sein. Die Koordinaten können nicht im **Move-**Befehl selber spezifiziert werden. Beschleunigung und Verzögerung für die **Move**-Anweisung werden durch den AccelS-Befehl gesteuert. Die Geschwindigkeit für die Bewegung wird durch den SpeedS-Befehl gesteuert. Wenn der SpeedS Geschwindigkeitswert die erlaubte Geschwindigkeit für eine beliebige Achse überschreitet, wird der Strom für alle vier Achsmotoren abgeschaltet und der Roboter hält an.

Die optionale Till-Bedingung gestattet es dem Anwender, eine Bedingung zu definieren, die den Roboter dazu veranlasst, bis zum vollständigen Stopp zu verzögern, bevor der **Move**-Befehl vollständig ausgeführt wurde. Die spezifizierte Bedingung ist ganz einfach eine Gegenprüfung zu einem der Eingänge. Dies wird durch Verwenden des Sw-Befehls erreicht. Der Anwender kann überprüfen, ob die Eingänge aus- oder eingeschaltet sind und den Arm dazu veranlassen, anzuhalten, je nachdem, welche Bedingung spezifiziert ist. Dieses Feature arbeitet ähnlich wie eine Unterbrechung (Interrupt), bei der **Move** unterbrochen (gestoppt) wird, sobald die Eingangs-Bedingung erfüllt ist. Wenn die Bedingung während des **Move**-Befehls nie erfüllt wird, erreicht der Arm erfolgreich den durch *destination* definierten Punkt. Für weitere Informationen bezüglich der Till-Bedingung lesen Sie bitte unter ‚Till-Befehl' weiter.

Hinweise

Was die Move-Anweisung nicht kann:

1) Move kann keine ausschließliche U-Achsen-Bewegung ausführen.

2) Move kann keine Bereichsüberprüfung der Bewegungsbahn durchführen, bevor die Bewegung selbst gestartet wird. Es ist dem System daher möglich, sogar für Zielpositionen, die sich innerhalb eines gestatteten Bereiches befinden, auf dem Weg zum Zielpunkt eine verbotene Position zu finden. In diesem Fall kann der Arm abrupt zum Stehen kommen, was einen Stoß und den Servo-Aus-Zustand des Arms hervorrufen kann. Um dem vorzubeugen, sollten Sie sicherstellen, dass bei niedriger Geschwindigkeit Bereichsüberprüfungen durchgeführt werden, bevor Sie **Move** bei hohen Geschwindigkeiten verwenden. Zusammenfassend heißt das, dass obschon sich die Zielposition innerhalb des Armbereiches befindet, es einige Bewegungen gibt, die nicht ausgeführt werden können. Das liegt daran, dass der Arm einige der Zwischenpositionen nicht erreichen kann, die während der **Move**-Bewegung benötigt werden.

Verwendung von Move mit CP

Der CP-Parameter veranlasst den Arm, zur *destination* zu fahren ohne langsamer zu werden oder an dem durch *destination* definierten Punkt anzuhalten. Dadurch wird dem Anwender ermöglicht, eine Serie von Bewegungsbefehlen miteinander zu verketten, was den Arm veranlasst, sich entlang eines kontinuierlichen Weges zu bewegen und während dieser Bewegung eine bestimmte Geschwindigkeit einzuhalten. Der **Move**-Befehl ohne CP veranlasst den Arm immer dazu, bevor er das Ziel *destination* erreicht hat, mit dem Herunterbremsen zu beginnen.

Move mit CP entspricht dem CMove auf den SRC 3xx Steuerungen.

Korrekte Speed (Geschwindigkeit) und Acceleration (Beschleunigung) Befehle mit Move:

Die Befehle SpeedS und AccelS werden verwendet, um Geschwindigkeit und Beschleunigung des Roboters während der **Move-**Bewegung zu definieren. Achten Sie darauf, dass SpeedS und AccelS sich auf linearinterpolierte und kreisinterpolierte Bewegungen beziehen, während eine PTP-Bewegung die Befehle Speed und Accel verwendet.

Potentielle Fehler

Versuch, nur die U-Achse zu betätigen

Da **Move** alle 4 Achsen im kartesischen Koordinatensystem steuert, ist es unmöglich, ausschließlich die U-Achse betätigen zu wollen. Wird dies versucht, tritt ein Fehler auf.

Fehler wegen Übergeschwindigkeit der Achse

Wenn eine angeforderte Bewegung ergibt , dass die Geschwindigkeit einer Achse überschritten wird, tritt ein Überdrehzahl-Fehler auf. Im Falle eines Überdrehzahl-Fehlers der Motoren wird der Roboterarm angehalten und die Servoachse wird abgeschaltet.

Es folgt kein Bewegungsbefehl auf den Move CP -Befehl

Wenn auf den **Move** CP –Befehl keine weiteren Bewegungsbefehle folgen, die den Arm reibungslos bewegen (da der Arm sich noch nicht verzögert hat), kann der Arm durch plötzliche Stoßbewegungen beschädigt werden. In diesem Falle tritt ein Fehler auf, der Arm hält an und die Servoachse wird freigeschaltet. (Eingeben einer Pause während einer CP-Bewegung kann ebenfalls einen Fehler hervorrufen, da der Befehl ebenfalls versucht, den Roboter zum sofortigen Stillstand zu bringen).

Verwandte Befehle

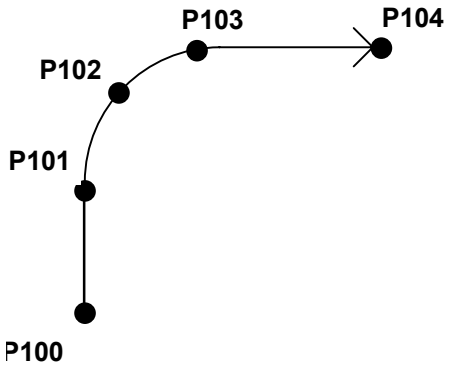
AccelS, Arc, Go, Jump, SpeedS, Sw, Till

Beispiel einer Move Anweisung

Das folgende Beispiel zeigt eine einfache PTP-Bewegung zwischen den Punkten P0 und P1. Anschließend bewegt sich der Arm geradlinig zu Punkt P2, bis Eingang #2 einschaltet. Wenn Eingang #2 während der Bewegung einschaltet, dann verzögert sich der Arm bis zum vollständigen Stopp, bevor er Punkt P2 erreicht und der nächste Programmbefehl ausgeführt wird.

```
Function movetest
    Home
    Go P0
    Go P1
    Move P2 Till Sw(2) = 1
    If Sw(2) = 1 Then GoTo ioset Else GoTo movefin
ioset:
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
    GoTo continue
movefin:
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
continue:
Fend
```

Dieses Beispiel verwendet **Move** mit CP. Das Diagramm unten zeigt eine Bogenbewegung, die am Punkt P100 ihren Anfang nimmt, in einer geraden Linie durch Punkt P101 läuft, an welchem der Arm beginnt, einen Bogen zu schlagen. Der Bogen wird dann fortgesetzt durch den Punkt P102 und weiter bis P103. Danach bewegt sich der Arm in einer geraden Linie zu Punkt P104, wo er sich dann endlich bis zum vollständigen Stopp verzögert. Bitte beachten Sie, dass sich der Arm zwischen den einzelnen Punkten nicht verzögert, bis er am Punkt P104 ankommt. Die folgende Funktion würde eine solche Bewegung generieren.




```
Function CornerArc
   Go P100
   Move CP P101       'Stoppt nicht bei P101
   Arc CP P102, P103  'Stoppt nicht bei P103
   Move P104          'Verzögert, um bei P104 zu stoppen
Fend
```

MsgBox Anweisung

> S

Zeigt eine Nachricht in einem Dialogfenster an und wartet darauf, dass der Anwender einen Button auswählt.

Syntax

MsgBox *msg$, type, title$, answer*

Parameter

msg$ Die Nachricht, die angezeigt wird.

type Ein numerischer Ausdruck, der die Summe von Werten darstellt, welche die Anzahl und den Typ der anzuzeigenden Buttons spezifizieren, den Stil der Icons und die Identität des Default-Buttons. EPSON RC+ schließt vordefinierte Konstanten mit ein, die für diesen Parameter verwendet werden können. Die folgende Tabelle zeigt die Werte an, die verwendet werden können.

Symbolische Konstante	Wert	Bedeutung
MB_OK	0	Ausschl. Anzeige des OK-Buttons.
MB_OKCANCEL	1	Anzeige der Buttons OK und Cancel.
MB_ABORTRETRYIGNORE	2	Anzeige der Buttons Abort, Retry und Ignore.
MB_YESNOCANCEL	3	Anzeige der Buttons Yes, No und Cancel.
MB_YESNO	4	Anzeige der Buttons Yes und No.
MB_RETRYCANCEL	5	Anzeige der Buttons Retry und Cancel.
MB_ICONSTOP	16	Stop-Zeichen.
MB_ICONQUESTION	32	Fragezeichen.
MB_ICONEXCLAMATION	64	Ausrufezeichen.
MB_DEFBUTTON1	0	Der erste Button entspricht der Vorgabeeinstellung.
MB_DEFBUTTON2	256	Der zweite Button entspricht der Vorgabeeinstellung.

title$ Zeichenkettenausdruck, der in der Titelzeile des Dialogfensters angezeigt wird.

answer Eine Integer-Variable, die einen Wert empfängt, welcher die vom Anwender gewählte Aktion angibt. EPSON RC+ schließt vordefinierte Konstanten mit ein, die für diesen Parameter verwendet werden können. Die Tabelle unten zeigt Werte an, die in *answer* ausgegeben werden.

Symbolische Konstante	Wert	Bedeutung
IDOK	1	OK-Button ausgewählt.
IDCANCEL	2	Cancel-Button ausgewählt.
IDABORT	3	Abort-Button ausgewählt.
IDRETRY	4	Retry-Button ausgewählt.
IDYES	6	Yes-Button ausgewählt.
IDNO	7	Kein Button ausgewählt.

Beschreibung

MsgBox formatiert die Nachricht automatisch. Wenn Sie Leerzeichen wünschen, verwenden Sie Chr$(13) + Chr$(10) in der Nachricht. Siehe Beispiel.

Verwandte Befehle

InputBox

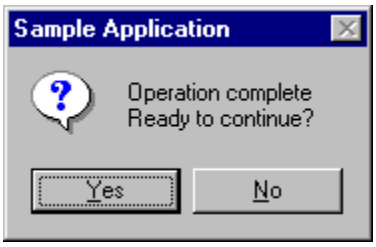
MsgBox Beispiel

Dieses Beispiel zeigt ein Nachrichtenfenster an, das den Operator fragt, ob er fortfahren möchte oder nicht. Das Nachrichtenfenster zeigt zwei Buttons an: *Yes* und *No* (Ja und Nein). Ein Fragezeichen-Icon wird ebenfalls angezeigt. Nach Return von MsgBox (nachdem der Anwender auf einen Button geklickt hat), wird die Antwort überprüft. Wenn sie auf *No* (Nein) lautet, werden alle Tasks mit dem Quit-Befehl beendet.

```
Function msgtest
   String msg$, title$
   Integer mFlags, answer

   msg$ = "Operation complete" + Chr$(13) + Chr$(10)
   msg$ = msg$ + "Ready to continue?"
   title$ = "Sample Application"
   mFlags = MB_YESNO + MB_ICONQUESTION
   MsgBox msg$, mFlags, title$, answer
   If answer = IDNO then
      Quit All
   EndIf
Fend
```

Ein Bild des Nachrichtenfensters, das dieser Code erzeugt, wird unten dargestellt.



MyTask Funktion

F

Gibt die Tasknummer des aktuellen Programmes aus.

Syntax

MyTask

Rückgabewerte

Die Tasknummer des aktuellen Tasks. Gültige Einträge sind die Integers 1-32.

Beschreibung

MyTask gibt die Tasknummer des aktuellen Programms mit einem Zahlzeichen aus. Der **MyTask** – Befehl wird innerhalb eines bestimmten Programmes eingefügt. Wenn dieses Programm die **MyTask-**Funktion ausführt, wird die Tasknummer ausgegeben, in der das Programm läuft.

Verwandte Befehle

Xqt

Beispiel einer MyTask Funktion

Das folgenden Programm schaltet die E/A Ports von 1 bis 8 Ein oder Aus.

```
Function main
  Xqt 2, task    'Führt Task 2 aus.
  Xqt 3, task    'Führt Task 3 aus.
  Xqt 4, task    'Führt Task 4 aus.
  Xqt 5, task    'Führt Task 5 aus.
  Xqt 6, task    'Führt Task 6 aus.
  Xqt 7, task    'Führt Task 7 aus.
  Xqt 8, task    'Führt Task 8 aus.
  Call task
Fend

Function task
  Do
    On MyTask    'Schaltet den E/A-Port ein, der dieselbe Nummer hat
                 'wie die aktuelle Tasknummer
    Off MyTask    'Schaltet den E/A-Port aus, der dieselbe Nummer hat
                 'wie die aktuelle Tasknummer
  Loop
Fend
```

Next

> S

Die Befehle For/Next werden zusammen verwendet, um eine Schleife zu erzeugen, in welcher Befehle, die sich zwischen den Befehlen For und Next befinden, mehrfach ausgeführt werden, wie vom Anwender angegeben.

Syntax

For *var1* **=** *initval* **To** *finalval* **[Step** *Increment* **]**
statements
Next *var1*

Parameter

var1	Die Zählvariable, die mit der For/Next Schleife verwendet wird. Diese Variable wird normalerweise als Integer definiert, kann jedoch auch als Realvariable definiert werden.
initval	Der Anfangswert für den Zähler *var1*.
finalval	Der Endwert des Zählers var1. Sobald dieser Wert erreicht ist, ist die For/Next Schleife vollständig und die Ausführung wird mit der Anweisung, welche auf den Next-Befehl folgt, fortgesetzt.
Increment	Ein optionaler Parameter, der das Zähl-Inkrement für jedes Mal definiert, welches die Next-Anweisung innerhalb der For/Next Schleife ausgeführt wird. Diese Variable kann positiv oder negativ sein. Wenn der Wert jedoch negativ ist, muss der Erstwert der Variable größer sein, als ihr Endwert. Wird der Inkrement-Wert weggelassen, inkrementiert (erhöht) das System automatisch um 1.
statements	Jede gültige SPEL Anweisung kann in die For/Next Schleife eingefügt werden.

Rückgabewerte

Keine

Beschreibung

For/Next führt einen Satz von Anweisungen innerhalb einer Schleife eine definierte Anzahl von Malen aus. Die *For* Anweisung stellt den Anfang der Schleife dar. Die **Next** Anweisung ist das Ende der Schleife. Die Male, welche die Anweisungen innerhalb der Schleife ausgeführt werden, werden mithilfe einer Variable gezählt.

Der erste numerische Ausdruck (*initval)* ist der Erstwert des Zählers. Dieser Wert kann positiv oder negativ sein, solange die Variable *finalval* und die Step-Inkrementierung einander korrekt entsprechen.

Der zweite numerische Ausdruck (*finalval)* ist der Endwert des Zählers. Dies ist der Wert der, sobald er erreicht ist, die Beendigung der For/Next Schleife auslöst, und die Steuerung des Programms wird an den nächsten auf den **Next**-Befehl folgenden Befehl weitergegeben.

Programmanweisungen, die der *For* Anweisung folgen, werden ausgeführt bis ein **Next**-Befehl erreicht wird. Die Zählervariable (*var1*) wird dann durch den Step-Wert inkrementiert, der durch den Parameter *increment* definiert ist. Wird die Step Option nicht genutzt, wird der Zähler um 1 inkrementiert.

Die Zählervariable (*var1*) wird dann mit dem Endwert (*finalval*) verglichen. Wenn der Zählerstand kleiner oder gleich dem Endwert (*finalval*) ist, werden die Anweisungen, die dem **For** Befehl folgen, erneut ausgeführt. Wenn die Zählervariable größer als der Endwert (*finalval*) ist, wird die Ausführung außerhalb der For/**Next** Schleife verzweigt und fährt mit dem Befehl fort, welcher direkt auf den **Next** Befehl folgt.

Eine Verschachtelung von For/**Next** Anweisungen wird bis zu einer Tiefe von 10 Leveln unterstützt. Dies bedeutet, dass eine For/**Next**-Schleife in einer weiteren For/**Next**-Schleife verschachtelt werden kann usw., bis 10 "Verschachtelungen" von For/**Next**-Schleifen vorhanden sind.

HINWEIS

NEGATIVE Step-Werte:
Wenn der Wert der Step-Inkrementierung (*increment*) negativ ist, wird die Zählervariable (*var1*) bei jedem Durchlauf durch die Schleife dekrementiert (verringert) und der Erstwert (*initval*) muss größer sein, als der Endwert, damit die Schleife funktioniert.

Verwandte Befehle
For

For/Next Beispiel

```
Function fornext
    Integer ctr
    For ctr = 1 to 10
        Go Pctr
    Next ctr
    '
    For ctr = 10 to 1 Step -1
        Go Pctr
    Next ctr
Fend
```

Not Operator

F

Führt die Bit-weise Umkehrung des Operandenwertes aus.

Syntax
Not o*perand*
Parameter
operand Integer-Ausdruck.

Rückgabewerte
Umkehrung des Operandenwertes.
Beschreibung
Die **Not**-Funktion führt die Bit-weise Umkehrung des Operandenwertes aus. Jedes resultierende Bit ist die Umkehrung des entsprechenden Bits im Operanden, mit tatsächlicher Änderung von 0-Bits nach 1, und 1er-Bits nach 0.

Verwandte Befehle
Abs, And, Atan, Atan2, Cos, Int, LShift, Mod, Or, RShift, Sgn, Sin, Sqr, Str$, Tan, Val, Xor

Beispiel des Not Operators

Dies ist ein einfaches Online-Fenster-Beispiel bezüglich der Verwendung des **Not**-Befehls.

```
>print not(1)
 -2
>
```

Off Anweisung

> S

Schaltet den angegebenen Ausgang aus, der nach einer definierten Zeit durch *Time* wieder eingeschaltet werden kann.

Syntax

Off { *bitNumber* | *outputLabel* } [, *time* [, *parallel*]]

Parameter

bitNumber Integer-Ausdruck, dessen Wert zwischen 0 -511 liegt. Sagt dem **Off-**Befehl, welcher Ausgang ausgeschaltet werden soll.

outputLabel Ausgangslabel.

time Optional. Gibt ein Zeitintervall in Sekunden an, für das der Ausgang ausgeschaltet bleiben soll. Nachdem das Zeitintervall abgelaufen ist, wird der Ausgang wieder eingeschaltet. (Das minimale Zeitintervall beträgt 0.01 Sekunden)

parallel Optional. Wenn ein Timer eingestellt ist, kann der *parallel*-Parameter dazu verwendet werden festzulegen, wann der nächste Befehl ausgeführt wird:

0 – sofort nachdem der Ausgang ausgeschaltet wurde

1 – nach Ablauf des definierten Zeitintervalls. (Vorgabewert)

Beschreibung

Off schaltet den angegebenen Ausgang aus (setzt ihn auf 0).

Wenn der *time*-Intervallparameter definiert ist, wird der durch *bitNumber* spezifizierte Ausgang ausgeschaltet, und nach Ablauf des Zeitintervalls wieder eingeschaltet. Wenn der Ausgang vor Auführung von Off bereits ausgeschaltet war, wird er nach Ablauf des Zeitintervalls wieder eingeschaltet.

Die *parallel*-Parametereinstellungen sind anwendbar, wenn das Zeitintervall wie folgt definiert ist:
1: Schaltet den Ausgang aus, nach Ablauf eines definierten Zeitintervalles wieder ein und führt dann den nächsten Befehl aus. (Dies ist auch der Vorgabewert für den *parallel*-Parameter. Wird dieser Parameter weggelassen, ist dies das gleiche, als würde der Parameter auf 1 gesetzt.)
0: Schaltet den Ausgang aus und führt gleichzeitig den nächsten Befehl aus.

Einschränkung

Ausgänge, die als Remote-Control-Ausgänge konfiguriert sind

Wenn ein Ausgang spezifiziert wird, der als Systemausgang eingestellt war, so tritt ein Fehler auf. Remote-Control-Ausgänge werden je nach Systemstatus automatisch ein- oder ausgeschaltet.

Verhalten der Ausgänge im Falle von Not-Halt:

EPSON RC+ hat ein Feature das veranlasst, dass bei Eintreten eines Not-Halt alle Ausgänge ausgeschaltet werden. Dieses Feature wird über den SPEL-Optionen-Tab im Systemkonfigurationsdialog eingestellt oder deaktiviert, zu dem man über das Setup-Menü gelangt.

Der Unterschied zwischen Off und Off $:

Es ist sehr wichtig, dass der Anwender den Unterschied zwischenden Befehlen **Off** und Off $ versteht.

Der Off $ -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf die Hardware E/As des Systems.

- Der **Off**-Befehl arbeitet mit den Standard- und mit der Erweiterungs-Hardware Ausgangsports. Diese Hardwareports sind getrennte Ausgänge, die mit Geräten kommunizieren, die außerhalb des Steuerungsgerätes liegen.

Verwandte Befehle

In, In$, InBCD, Off$, On$, OpBCD, Oport, Out, Out$, Sw$, Wait

Beispiel einer Off Anweisung

Das Beispiel unten zeigt, wie der Haupttask einen Hintergrundtask namens *iotask* startet. *iotask* ist ein einfacher Task, um die getrennten Ausgänge 1 und 2 ein- und danach wieder auszuschalten, 10 Sekunden zu warten und den Vorgang dann zu wiederholen.

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
  .
Fend


Function iotask
  Do
    On 1
    On 2
    Off 1
    Off 2
    Wait 10
  Loop
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> On  1
> Off  1,10      'Schaltet den Ausgang 1 aus, wartet 10 Sekunden ab und
                 schaltet ihn wieder ein.
> On 2
> Off 2
```

Off $ Anweisung

> S

Schaltet das spezifizierte Bit des S/W Merkers aus.

Syntax

Off ${ *bitNumber* | *memIOLabel* }

Parameter

bitNumber Integer-Ausdruck zwischen 0-511, der einen der 512 S/W Merker repräsentiert. HINWEIS: Das Dollarzeichen "$" muss vor Bitnummer oder Label gesetzt werden um zu verdeutlichen, dass es sich hierbei um einen Merker und nicht um einen Hardware-Ausgang handelt.

memIOLabel Merker-Label.

Beschreibung

Off $ schaltet das definierte Bit des Merkers aus (setzt es auf 0). Die 512 Merker-Bits sind normalerweise eine gute Wahl für die Verwendung als Status-Bits für Zwecke wie Ein/Aus, Wahr/Falsch, Erledigt/Nicht Erledigt etc. Der Befehl On $ schaltet den Merker ein, während **Off $** ihn ausschaltet. Der Befehl Sw $ wird verwendet, um den aktuellen Status den angegebenen Merkers zu überprüfen. Der Befehl Wait kann ebenfalls mit dem Merker verwendet werden, um das System dazu zu veranlassen zu warten, bis ein definierter S/W-Status eingestellt ist.

Einschränkung

Der Unterschied zwischen Off und Off $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischenden Befehlen Off und **Off $** versteht.

Der **Off $** -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf die Hardware E/As.

- Der **Off**-Befehl arbeitet mit den Standard- und mit der Erweiterungs-Hardware Ausgangsports. Diese Hardwareports sind einzelne Ausgänge, die mit Geräten außerhalb der Steuerung kommunizieren.

Verwandte Befehle

In, In$, InBCD, Off, On, On$, OpBCD, Oport, Out, Out$, Sw, Sw$, Wait

Beispiel einer Off$ Anweisung

Das folgende Beispiel zeigt 2 Tasks. Jeder der beiden Tasks hat die Möglichkeit, Bewegungsbefehle zu initiieren. Jedoch wird ein Sicherungsmechanismus zwischen den beiden Tasks verwendet um sicherzustellen, dass ein Task erst dann die Kontrolle über die Bewegungsbefehle des Roboters erhält, wenn der andere Task deren Verwendung abgeschlossen hat. Dies ermöglicht es 2 Tasks, Bewegungsbefehle korrekt und in geordneter, vorhersehbarere Art und Weise, auszuführen. Sw $ wird in Kombination mit dem Wait-Befehl verwendet, um zu warten bis der Merker Nr. 1 den richtigen Wert erreicht hat, von dem an es sicher ist, eine neue Bewegung auszuführen. On $ und Off $ werden verwendet, um den S/W-Merker für die richtige Synchronisierung ein- oder auszuschalten.

```
Function main
    Integer I
    Off $1
    Xqt 2, task2
    For I = 1 to 100
        Wait Sw($1) = 0
        Go P(i)
        On $1
    Next I
Fend
```

```
Function task2
    Integer I
    For I = 101 to 200
        Wait Sw($1) = 1
        Go P(i)
        Off $1
    Next I
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> On  $1      'Schaltet den Merker #1 ein
> Print Sw($1)
1
> Off $1    'Schaltet den Merker #1 aus
> Print Sw($1)
0
```

OLRate Anweisung

> S

Anzeige des Überlastgrades für ein oder alle Achsen des aktuellen Roboters. Gibt die OLRate für die angegebene Achse aus.Die Werte variieren zwischen 0.0 und 2.0.
.

Syntax

OLRate [*jointNumber*]

Parameter

jointNumber Integer-Ausdruck, dessen Wert zwischen 1 und 4 liegt.

Beschreibung

OLRate kann verwendet werden, um festzustellen, ob ein bestimmter Zyklus das Servosystem überbeansprucht. Faktoren wie Temperatur und Strom können Servofehler während der Anwendung in Hochleistungszyklen verursachen. **OLRate** kann dabei helfen zu überprüfen, ob das Robotersystem kurz vor einem Servofehler steht.

Während eines Zyklus' sollten Sie einen weiteren Task ausführen, um **OLRate** zu überwachen. Wenn die **OLRate** 1.0 für eine beliebige Achse überschreitet, so tritt ein Servofehler auf.

Servofehler treten am ehesten bei hohen Nutzlasten auf. Durch die Verwendung von OLRate in einem Testzyklus können Sie sicherstellen, dass durch die Geschwindigkeits- und Beschleunigungseinstellungen kein Servofehler im Produktionsmodus auftritt.

Um gültige Messwerte zu erhalten müssen Sie **OLRate** ausführen während sich der Roboter bewegt.

Verwandte Befehle

OLRate Funktion

Beispiel einer OLRate Anweisung

```
>olrate
0.10000  0.20000
0.30000  0.40000
0.50000  0.60000

Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Do
    ' Display OLRate
    OLRate
    Wait 1
  Loop
Fend
```

OLRate Funktion

F

Gibt den Überlastgrad für eine Achse des aktuellen Roboters aus.

Syntax

OLRate(*jointNumber***)**

Parameter

jointNumber Integer-Ausdruck, dessen Wert zwischen 1 und 4 liegt.

Rückgabewerte

Gibt die OLRate für die angegebene Achse aus. Die Werte variieren zwischen 0.0 und 2.0.

Beschreibung

OLRate kann verwendet werden, um festzustellen, ob ein bestimmter Zyklus das Servosystem überbeansprucht. Faktoren wie Temperatur und Strom können Servofehler während der Anwendung in Hochleistungszyklen verursachen. **OLRate** kann dabei helfen zu überprüfen, ob das Robotersystem kurz vor einem Servofehler steht.

Während eines Zyklus' sollten Sie einen weiteren Task ausführen, um **OLRate** zu überwachen. Wenn die **OLRate** 1.0 für eine beliebige Achse überschreitet, so tritt ein Servofehler auf.

Servofehler treten am ehesten bei hohen Nutzlasten auf. Durch die Verwendung von OLRate in einem Testzyklus können Sie sicherstellen, dass durch die Geschwindigkeits- und Beschleunigungseinstellungen kein Servofehler im Produktionsmodus auftritt.

Um gültige Messwerte zu erhalten, müssen Sie **OLRate** ausführen während sich der Roboter bewegt.

Verwandte Befehle

OLRate Anweisung

Beispiel einer OLRate Funktion

```
Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Integer i
  Real olRates(4)
  Do
    For i = 1 to 4
      olRates(i) = OLRate(i)
      If olRate(i) > .5 Then
        Print "Warning: OLRate(", i, ") is over .5"
      EndIf
    Next i
  Loop
Fend
```

On Anweisung

> S

Schaltet den angegebenen Ausgang ein, der nach einer definierten Zeit durch *Time* wieder ausgeschaltet werden kann.

Syntax

On { *bitNumber* | *outputLabel* } [, *time* [, *parallel*]]

Parameter

bitNumber Integer-Ausdruck, dessen Wert zwischen 0 -511 liegt. Sagt dem **On-**Befehl, welcher Ausgang eingeschaltet werden soll.

outputLabel Ausgangslabel.

time Optional. Gibt ein Zeitintervall in Sekunden an, für das der Ausgang eingeschaltet bleiben soll. Nachdem das Zeitintervall abgelaufen ist, wird der Ausgang wieder ausgeschaltet. (Das minimale Zeitintervall beträgt 0.01 Sekunden)

parallel Optional. Wenn ein Timer eingestellt ist, kann der *parallel*-Parameter dazu verwendet werden festzulegen, wann der nächste Befehl ausgeführt wird:

0 – sofort nachdem der Ausgang eingeschaltet wurde

1 – nach Ablauf des definierten Zeitintervalls. (Vorgabewert)

Beschreibung

On schaltet den angegebenen Ausgang ein (setzt ihn auf 1).
Wenn der Zeitintervall-Parameter definiert ist, wird der durch *outnum* spezifizierte Ausgang eingeschaltet, und nach Ablauf des Zeitintervalls wieder ausgeschaltet.

Die *parallel*-Parametereinstellungen sind anwendbar, wenn das Zeitintervall wie folgt definiert ist:
1: Schaltet den Ausgang ein, nach Ablauf eines definierten Zeitintervalles wieder aus und führt dann den nächsten Befehl aus. (Dies ist auch der Vorgabewert für den *parallel*-Parameter. Wird dieser Parameter weggelassen, ist dies das gleiche, als würde der Parameter auf 1 gesetzt.)
0: Schaltet den Ausgang ein und führt gleichzeitig den nächsten Befehl aus.

Einschränkung

Ausgänge, die als Remote konfiguriert sind

Wird ein Ausgang spezifiziert, der als Remote eingestellt war, so tritt ein Fehler auf. Remote-Ausgänge werden je nach Systemstatus automatisch ein- oder ausgeschaltet. Für weitere Remote-Details lesen Sie bitte das EPSON RC+ Benutzerhandbuch. Die einzelnen Bits für den Remote-Connector können vom über das Setup-Menü erreichbaren EPSON RC+ Remote-Konfigurationsdialog als Remote oder als E/A eingestellt werden.

Verhalten der Ausgänge im Falle von Not-Halt

EPSON RC+ hat ein Feature das veranlasst, dass bei Eintreten eines Not-Halt alle Ausgänge ausgeschaltet werden. Dieses Feature wird über einen der Option-Schalter eingestellt oder deaktiviert. Um das Feature zu konfigurieren, gehen Sie bitte auf den SPEL-Options-Tab im System-Konfigurationsdialog, den Sie über das Setup Menü erreichen können.

Der Unterschied zwischen On und On $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischenden Befehlen **On** und On $ versteht.

Der On $ -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf die Hardware E/As.

- Der **On**-Befehl arbeitet mit den Standard- und mit der Erweiterungs-Hardware Ausgangsports. Diese Hardwareports sind einzelne Ausgänge, die mit Geräten außerhalb der Steuerung kommunizieren.

Verwandte Befehle

In, In $, InBCD, Off, Off $, On $, OpBCD, Oport, Out, Out $, Sw$, Wait

Beispiel einer On Anweisung

Das Beispiel unten zeigt, wie der Haupttask einen Hintergrundtask namens *iotask* startet. *iotask* ist ein einfacher Task, um die getrennten Ausgänge 1 und 2 ein- und danach wieder auszuschalten, 10 Sekunden zu warten und den Vorgang dann zu wiederholen.

```
Function main
    Xqt iotask
    Go P1
    .
    .
    .
Fend


Function iotask
    '
    Do
        On 1
        On 2
        Off 1
        Off 2
        Wait 10
    Loop
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> On 1
> Off 1
> On 2
> Off 2
```

On $ Anweisung

> S

Schaltet das spezifizierte Bit des S/W Merkers ein.

Syntax

On $*bitNumber*

Parameter

bitNumber Zahl zwischen 0 und 511, die einen der 512 Merker darstellt. HINWEIS: Das Dollarzeichen "$" muss vor Bitnummer oder Label gesetzt werden um zu verdeutlichen, dass es sich hierbei um einen Merker und nicht um einen Hardware-Ausgang handelt.

Beschreibung

On $ schaltet das definierte Bit des Roboter-Merkers ein (setzt es auf 1). Die 512 Merker-Bits sind normalerweise eine gute Wahl für die Verwendung als Status-Bits für Zwecke wie Ein/Aus, WAHR/FALSCH, ERLEDIGT/Nicht ERLEDIGT etc. Der Befehl On $ schaltet den Merker ein, während Off $ ihn ausschaltet. Der Befehl Sw $ wird verwendet, um den aktuellen Status den angegebenen Merkers zu überprüfen. Der Befehl Wait kann ebenfalls mit dem Merker verwendet werden, um das System dazu zu veranlassen zu warten, bis ein definierter S/W-Status eingestellt ist.

Einschränkung

Der Unterschied zwischen On und On $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischenden Befehlen On und **On $** versteht.

- Der **On $** -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf die Hardware E/As des Roboters.
- Der **On**-Befehl arbeitet mit den Standard- und mit der Erweiterungs-Hardware Ausgangsports. Diese Hardwareports sind einzelne Ausgänge, die mit Geräten außerhalb der Steuerung kommunizieren.

Verwandte Befehle

In, In$, InBCD, Off, Off$, On, OpBCD, Oport, Out, Out$, Sw, Sw$, Wait

Beispiel einer On$ Anweisung
Das folgende Beispiel zeigt 2 Tasks. Jeder der beiden Tasks hat die Möglichkeit, Bewegungsbefehle zu initiieren. Jedoch wird ein Sicherungsmechanismus zwischen den beiden Tasks verwendet um sicherzustellen, dass ein Task erst dann die Kontrolle über die Bewegungsbefehle des Roboters erhält, wenn der andere Task deren Verwendung abgeschlossen hat. Dies ermöglicht es 2 Tasks, Bewegungsbefehle korrekt und in geordneter, vorhersehbarer Art und Weise, auszuführen. Sw $ wird in Kombination mit dem Wait-Befehl verwendet, um zu warten bis Merker Nr. 1 den richtigen Wert erreicht hat, von dem an es sicher ist, eine neue Bewegung auszuführen. On $ und Off $ werden verwendet, um den S/W-Merker für die richtige Synchronisierung ein- oder auszuschalten.

```
Function main
    Integer I
    Off $1
    Xqt 2, task2
    For I = 1 to 100
        Wait Sw($1) = 0
        Go P(i)
        On $1
    Next I
Fend


Function task2
    Integer I
    For I = 101 to 200
        Wait Sw($1) = 1
        Go P(i)
        Off $1
    Next I
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> on $1
> print sw($1)
1
> off $1
> print sw($1)
0
```

OnErr

S

Legt Interrupt-Verzweigungen an, welche die Steuerung veranlassen, zu einer Fehlerbehandlungs-Subroutine zu wechseln, wenn ein Fehler auftritt. Gestattet dem Anwender die Fehlerbehandlung.

Syntax

OnErr GoTo {*label* | *lineNumber* | **0**}

Parameter

lineNumber	Anweisungszeilennummer die ausgeführt werden soll, wenn ein Fehler auftritt.
label	Anweisungslabel zu dem gewechselt werden soll, wenn ein Fehler auftritt.
0	Der Parameter, der verwendet wird, um die OnErr Einstellung zu löschen.

Beschreibung

OnErr gestattet dem Anwender die Fehlerbehandlung. Wenn ein Fehler auftritt, ohne dass **OnErr** verwendet wird, wird der Task abgebrochen und der Fehler wird angezeigt. Wenn **OnErr** jedoch verwendet wird, gestattet dies dem Anwender, den Fehler "abzufangen" und zu einer Fehlerbehandlungsroutine zu gehen, um den Fehler automatisch zu beheben. Bei Empfang eines Fehlers verzweigt **OnErr** die Steuerung zur im **OnErr**-Befehl designierten Zeilennummer oder dem entsprechenden Label. Auf diese Art und Weise wird der Task nicht abgebrochen und dem Anwender wird die Möglichkeit gegeben, den Fehler automatisch zu beheben. Dies macht den Arbeitsablauf in Roboter-Arbeitszellen wesentlich reibungsloser, da potentielle Probleme immer auf die gleiche Weise behandelt und behoben werden.

Während der Ausführung der Fehlerbehandlungsroutine (ein Fehlerbehandlungs-Unterprogramm) muss der Fehler behoben werden, damit die Steuerung in das Hauptprogramm zurückkehren kann. Dies wird mit dem EClr-Befehl erreicht. (Siehe unten stehende Einschränkungen.) Sobald der EClr-Befehl den Fehler gelöscht hat, kann die Steuerung zu der Anweisungs-Zeile zurückkehren, welche direkt auf diejenige Zeile folgt, in welcher der Fehler zuerst auftrat.

Wenn der **OnErr** Befehl mit dem 0 Parameter spezifiziert wird, wird die aktuelle **OnErr** Einstellung gelöscht. (D.h. nach der Ausführung von **OnErr** 0 wird die Programmausführung unterbrochen wenn ein Fehler auftritt).

Tipps

Platzierung des OnErr Befehls:

Bedenken Sie bitte, dass die Fehlerbehandlung deaktiviert ist, bis der **OnErr** Befehl ausgeführt wurde. Daher ist es normalerweise eine gute Idee, den **OnErr** Befehl an den Anfang eines Programms zu setzen, damit Fehler von Programmbeginn an automatisch bearbeitet werden können.

Einschränkungen

Denken Sie daran, dass alle Fehler mit der EClr-Anweisung gelöscht werden müssen:
Wenn der Anwender in der Fehlerbehandlungsroutine jedoch keine EClr Anweisung verwendet, um den Fehler zu löschen, kann das System in einer Endlosschleife stecken bleiben. Wenn der Fehler nicht mithilfe der Fehlerbehandlungsroutine zurückgesetzt wird, gibt das System auf den **OnErr**-Befehl hin die Steuerung sofort nach seiner Rückkehr zur Hauptroutine an das Fehlerbearbeitungsprogramm zurück. Der Grund dafür ist, dass das System davon ausgeht, es sei erneut ein Fehler aufgetreten, während es tatsächlich nur auf den vorangegangenen Fehler reagiert (der nicht gelöscht wurde). Da die Fehlerbehandlungsroutine den Fehler nicht zurücksetzt, tritt eine Endlosschleife auf. (Seien Sie daher vorsichtig!)

Verschachtelte Fehlerbehandlung
SPEL unterstützt keine verschachtelte Fehlerbehandlung. Das bedeutet, dass wenn beim Beheben eines Fehlers ein neuer Fehler auftritt, das System nicht in der Lage ist, erneut zur Fehlerbehandlungsroutine zu wechseln, um den zweiten Fehler zu beheben. In diesem Fall bricht das System den Task einfach ab und gibt eine Fehlermeldung aus.

Verwandte Befehle
EClr, Err

OnErr Beispiel

Das folgende Beispiel zeigt ein einfaches Dienstprogramm, das prüft, ob ob die Punkte P0-P399 existieren. Wenn der Punkt nicht existiert, erscheint eine Nachricht auf dem Bildschirm, um den Anwender diesen Umstand wissen zu lassen. Das Programm verwendet den CX-Befehl, um jeden einzelnen Punkt dahin gehend zu testen, ob er definiert wurde, oder nicht. Wenn ein Punkt nicht definiert ist, wird die Steuerung an die Fehlerbehandlungsroutine übergeben und auf dem Bildschirm erscheint eine Nachricht, die dem Anwender mitteilt, welcher Punkt nicht definiert war.

```
Function errDemo
    Integer i, errNum

    OnErr GoTo errHandler

    For i = 0 To 399
        temp = CX(P(i))
    Next i
    Exit Function
    '
    '
    '*****************************************
    '* Error Handler                         *
    '*****************************************
errHandler:
    errNum = Err(0)
    EClr
    'Überprüft, ob ein undefinierter Punkt verwendet wird
    If errNum = 78 Then
       Print "Point number P", i, " is undefined!"
    Else
       Print "ERROR: Error number ", errNum, " occurred while"
       Print "       trying to process point P", i, " !"
    EndIf
    EResume
Fend
```

OpBCD Anweisung

> S

Setzt 8 Ausgänge gleichzeitig unter Verwendung des BCD-Formats. (Binär codierte Dezimalstelle)

Syntax

OpBCD portNumber, outData

Parameter

portNumber Integer-Ausdruck, dessen Wert zwischen 0 und 63 liegt und jeweils eine der 64 Ausgangsgruppen repräsentiert (jede Gruppe enthält 8 Ausgänge), die die Standard- und die erweiterten Ausgänge des Systems bilden. Die *portNumber*-Auswahl entspricht den folgenden Ausgängen:

PortNummer	Ausgänge
0	0-7
1	8-15
2	16-23
3	24-31
...	...
63	504-511

outData Integer-Ausdruck zwischen 0 und 99, der das Ausgangsmuster für die mit der Anweisung *portNumber* ausgewählte Ausgangsgruppe repräsentiert. Die 2. Ziffer (Einer-Ziffer genannt) wird repräsentiert durch die unteren 4 Ausgänge in der ausgewählten Gruppe und die 1. Ziffer (Zehner-Ziffer genannt) wird repräsentiert durch die oberen 4 Ausgänge in der ausgewählten Gruppe.

Beschreibung

OpBCD setzt 8 Ausgänge gleichzeitig unter Verwendung des BCD-Formats. Die Standard- und Erweiterungs-Ausgänge werden in Gruppen von 8 unterteilt. Der *portNumber*-Parameter für den OpBCD-Befehl definiert, welche 8er-Ausgangsgruppe verwendet werden soll, wobei *portNumber* = 0 die Ausgänge 0-7 bedeutet , *portNumber* = 1 bedeutet die Ausgänge 8-15, etc.

Sobald eine Portnummer ausgewählt wurde (d.h. es wurde eine Gruppe von 8 Ausgängen ausgewählt), muss ein spezifisches Ausgangsmuster definiert werden. Dies wird im BCD-Format unter zuhilfenahme des *outdata* Parameters getan. Der *outdata* Parameter kann 1 oder 2 Ziffern haben. (Gültige Einträge liegen im Bereich von 0 bis 99.) Die erste Stelle (oder die Zehnerstelle) entspricht den oberen 4 Ausgängen der Gruppe von 8 Ausgängen, ausgewählt von *portNumber*. Die zweite Stelle (oder die Einerstelle) entspricht den unteren 4 Ausgängen der Gruppe von 8 Ausgängen, ausgewählt von *portNumber*.

Da gültige Ausgänge im BCD-Format im Bereich von 0 bis 9 für jede Stelle liegen, kann nicht jede E/A Kombination ausgeben werden. Die Tabelle unten zeigt einige der möglichen E/A Kombinationen und ihre entsprechenden *outnum*-Werte, davon ausgehend, dass die *portNumber* 0 ist.

Einstellungen des Ausgangs (Ausgangsnummer)

Outnum Wert	7	6	5	4	3	2	1	0
01	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
02	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)
03	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
08	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)
09	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)
10	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)
11	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
99	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)

Bitte beachten Sie, dass das BCD-Format nur die Spezifizierung von Dezimalwerten zulässt. Dies bedeutet, dass es durch die Verwendung des BCD-Formats nicht möglich ist, alle Ausgänge mit dem **OpBCD** Befehl einzuschalten. Bitte beachten Sie, dass der Maximalwert für jede *outnum*-Ziffer 9 ist. Dies bedeutet, dass der größte Wert, den man mit OpBCD verwenden kann, 99 ist. Aus der Tabelle oben ist leicht ersichtlich, dass 99 nicht alle Ausgänge einschaltet. Stattdessen schaltet 99 die Ausgänge 0, 3, 4, und 7 ein und alle anderen aus.

Einschränkung

Der Unterschied zwischen OpBCD und Out

Die Befehle **OpBCD** und Out sind einander in der SPEL Sprache sehr ähnlich. Es gibt jedoch einen Hauptunterschied zwischen den beiden. Dieser Unterschied wird im Folgenden aufgezeigt.

- Der **OpBCD** Befehl verwendet das BCD-Format zur Spezifizierung eines 8-Bit-Wertes der verwendet wird, um die Ausgänge ein- oder auszuschalten. Da das BCD-Format den Gebrauch der Werte &HA, &HB, &HC, &HD, &HE oder &HF ausschließt, können nicht alle Kombinationen für die Einstellung der 8 Ausgänge erfüllt werden.
- Der Out Befehl funktioniert ähnlich dem OpBCD Befehl, mit der Ausnahme, dass der Out Befehl es gestattet, den Bereich der 8-Bit-Werte, der für das Ein- oder Ausschalten der Ausgänge verwendet werden darf zwischen 0 und 255 anzusiedeln (gegenüber 0 bis 99 für OpBCD). Dies erlaubt es, alle möglichen Kombinationen für die 8 Ausgangsgruppen den Anwender-Spezifikationen folgend zu initialisieren.

Ausgänge, die als Remote konfiguriert sind:

Wird ein als Remote eingestellter Ausgang so spezifiziert, dass er durch **OpBCD** eingeschaltet wird, so tritt ein Fehler auf. Remote-Ausgänge werden je nach Systemstatus automatisch ein- oder ausgeschaltet. Für weitere Remote-Details lesen Sie bitte das EPSON RC+ Benutzerhandbuch. Die einzelnen Bits für den Remote-Connector können vom über das Setup-Menü erreichbaren EPSON RC+ Remote-Konfigurationsdialog als Remote oder als E/A eingestellt werden.

Verhalten der Ausgänge im Falle von Not-Halt:

EPSON RC+ hat ein Feature das veranlasst, dass bei Eintreten eines Not-Halt alle Ausgänge ausgeschaltet werden. Dieses Feature wird über einen der Option-Schalter aktiviert oder deaktiviert. Um das Feature zu konfigurieren, gehen Sie bitte auf den SPEL-Options-Tab im System-Konfigurationsdialog, den Sie über das Setup Menü erreichen können.

Verwandte Befehle

In, In $, InBCD, Off,Off $, On, On $, Oport, Out, Out $, Sw, Sw $, Wait

Beispiel einer OpBCD Funktion

Das Beispiel unten zeigt, wie der Haupttask einen Hintergrundtask namens *iotask* startet. *iotask* ist ein einfacher Task, um abwechselnd E/A Nrn. 1 & 2 ein- und dann E/A Nrn. 0 und 3 einzuschalten. Wenn 1 & 2 eingeschaltet sind, sind 0 & 3 ausgeschaltet und umgekehrt.

```
Function main
    Xqt 2, iotask
    Go P1
    .
    .
    .
Fend


Function iotask
    Do
        OpBCD 0,6
        OpBCD 0,9
        Wait 10
    Loop
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> OpBCD 1,6  'Schaltet die Ausgänge 1 & 2 ein.
> OpBCD 2,1  'Schaltet den Ausgang 8 ein.
> OpBCD 3,91 'Schaltet die Ausgänge 24, 28 & 31 ein
```

OpenCom Anweisung

S

Öffnen einer RS-232 Kommunikationsschnittstelle.

Syntax

OpenCom #*portNum*

Parameter

portNum Integer-Ausdruck für die zu öffnende Schnittstellennummer. Die Werte liegen zwischen 1 und 16.

Beschreibung

Eine RS-232 Schnittstelle muss aktiviert werden, bevor sie geöffnet werden kann. Siehe Kommunikationsschnittstellen-Tabulator: Konfiguration im Controller-Menü.

Verwandte Befehle

ChkCom, CloseCom, SetCom

Beispiel einer OpenCom Anweisung

```
OpenCom #1
```

OpenNet Anweisung

S

Öffnen eines TCP/IP Netzwerkports.

Syntax

OpenNet #*portNumber* **As** { **Client** | **Server** }

Parameter

portNumber Integer-Ausdruck für die zu öffnende Schnittstellennummer. Der Bereich liegt zwischen 128 und 131.

Beschreibung

OpenNet öffnet einen TCP/IP Port für die Kommunikation mit einem anderen Rechner im Netzwerk.

Ein System sollte als Server öffnen und das andere als Client. Es ist ohne Bedeutung, welches zuerst ausgeführt wird

Verwandte Befehle

ChkNet, CloseNet, SetNet

Beispiel einer OpenNet Anweisung

In diesem Beispiel haben zwei PCs ihre TCP/IP-Einstellungen wie folgt konfiguriert:

```
PC #1:
Port: #128
Host Name: PC2
TCP/IP Port: 1000

    Function tcpip1
      OpenNet #128 As Server
      WaitNet #128
      Print #128, "Data from host 1"
    Fend


    PC #2:
    Port: #128
    Host Name: PC1
    TCP/IP Port: 1000

    Function tcpip2
      String data$
      OpenNet #128 As Client
      WaitNet #128
      Input #128, data$
      Print "received '", data$, "' from host 1"
    Fend
```

Oport Funktion

F

Gibt den Status des spezifizierten Ausgangs aus.

Syntax

Oport(*outnum***)**

Parameter

outnum Zahl zwischen 0 und 511, die einen einzelnen Standard- oder einzelnen Erweiterungs-Ausgang repräsentiert.

Rückgabewerte

Gibt den angegebenen Ausgangsstatus als entweder 0 oder 1 aus.

0: Off (Aus) Status
1: On (Ein) Status

Beschreibung

Oport stellt einen Status-Check für die Ausgänge zur Verfügung. Er funktioniert weitgehend so, wie der Sw-Befehl für die Ausgänge funktioniert. **Oport** wird im allgemeinen verwendet, um den Status eines Ausganges zu überprüfen, welcher an einen Feeder, einen Conveyor, einen Greifermagneten oder an den Host eines anderen Gerätes angeschlossen sein sollte, welches über getrennte E/A arbeitet. Offensichtlich hat der Ausgang, der mit dem **Oport**-Befehl überprüft wird, 2 Stati (1 oder 0). Diese zeigen an, ob der spezifizierte Ausgang ein- oder ausgeschaltet ist.

Einschränkung

Der Unterschied zwischen Oport und Sw

Es ist sehr wichtig, dass der Anwender den Unterschied zwischen den Befehlen **Oport** und Sw versteht. Beide Befehle werden verwendet, um den E/A-Status zu erhalten. Der E/A-Typ ist jedoch bei Beiden unterschiedlich. Der Sw-Befehl arbeitet mit Eingängen. Der **Oport**-Befehl arbeitet mit den Standard- und mit Erweiterungs-Hardware-Ausgängen. Diese Hardwareports sind einzelne Ausgänge, die mit Geräten außerhalb der Steuerung kommunizieren.

Verwandte Befehle

In, In $, InBCD, Off, Off$, On, On $, OpBCD, Out, Out $, Sw, Sw $, Wait

Beispiel einer OPort Funktion

Das Beispiel unten schaltet Ausgang 5 ein und stellt dann sicher, *dass* dieser eingeschaltet ist, bevor es fortfährt.

```
Function main
    TMOut 10
    OnErr errchk
    Integer errnum
    On 5    'Schaltet Ausgang 5 ein
    Wait Oport(5)
    Call mkpart1
    Exit Function

errchk:
    errnum = Err(0); EClr
    If errnum = 94 Then
       Print "TIME Out Error Occurred during period"
       Print "waiting for Oport to come on. Check"
       Print "Output #5 for proper operation. Then"
       Print "restart this program."
    Else
       Print "ERROR number ", errnum, "Occurred"
       Print "Program stopped due to errors!"
    EndIf
    Exit Function
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> On 1
> Print Oport(1)
1
> Off 1
> Print Oport(1)
0
>
```

Or Operator

> S

Führt die bitweise Or-Operation (Oder-Verknüpfung) zwischen den Werten der Operanden aus.

Syntax

Operand **Or** *Operand*

Parameter

Operand Integer-Ausdruck.

Rückgabewerte

Bitweise or-verknüpfter Wert der Operanden.

Beschreibung

Der **ODER-Operator** führt die bitweise **Or**-Operation (Oder-Verknüpfung) zwischen den Werten der Operanden aus. Jedes Bit des Ergebnisses ist der **or**-verknüpfte Wert zwei einander entsprechender Bits der beiden Operanden.

Verwandte Befehle

And, LShift, Mod, Not, RShift, Xor

Beispiel des Or Operatoren

Dies ist ein einfaches Online-Fenster-Beispiel bezüglich der Verwendung des EXKLUSIV-ODER-Operatoren.

```
>print 1 or 2
 3
>
```

Out Anweisung

> S

Setzt 8 Ausgänge gleichzeitig.

Syntax

Out *portNumber, outData*

Parameter

portNumber Integer-Ausdruck zwischen 0 und 63, der jeweils eine der 64 Ausgangsgruppen repräsentiert (jede Gruppe enthält 8 Ausgänge), die die Standard- und die erweiterten Ausgänge bilden. Die *portnum*-Auswahl entspricht den folgenden Ausgängen:

Portnummer	Ausgänge
0	0-7
1	8-15
2	16-23
3	24-31
...	...
63	504-511

outData Integer zwischen 0 und 255, der das Ausgangsmuster für die mit der Anweisung *portNumber* ausgewählte Ausgangsgruppe repräsentiert. Wenn sie in hexadezimaler Form dargestellt wird, wird der Bereich von &H0 bis &HFF abgedeckt. Die niedrigere Ziffer repräsentiert die niederwertigen Stellen (oder die ersten 4 Ausgänge) und die obere Ziffer repräsentiert die höherwertigen Stellen (oder die zweiten 4 Ausgänge).

Beschreibung

Out setzt unter Nutzung der vom Anwender angegebenen Kombination aus *portNumber* und *outData* Werten gleichzeitig 8 Ausgänge. Die Ausgänge werden in 16 Gruppen zu je 8 Eingängen unterteilt. Der *portNumber*-Parameter für den OpBCD-Befehl definiert, welche 8er-Ausgangsgruppe verwendet werden soll, wobei *portNumber* = 0 die Ausgänge 0-7 bedeutet , *portNumber* = 1 bedeutet die Ausgänge 8-15, etc.

Sobald eine Portnummer ausgewählt wurde (d.h. es wurde eine Gruppe von 8 Ausgängen ausgewählt), muss ein spezifisches Ausgangsmuster definiert werden. Dafür wird der Parameter *outData* verwendet. Der *outData* Parameter kann einen Wert zwischen 0 und -255 haben und kann in hexadezimalem oder Integer Format repräsentiert werden. (D.h. &H0-&HFF oder 0-255)

Die Tabelle unten zeigt einige der möglichen E/A Kombinationen und ihre entsprechenden *outData*-Werte, davon ausgehend, dass die *portNumber* 0 oder entsprechend 1 ist.

Ausgangseinstellungen wenn *portNumber*=0 (Ausgangsnummer)

OutData Wert	7	6	5	4	3	2	1	0
01	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
02	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)
03	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
08	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)
09	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)
10	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)
11	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
99	Off (Aus)	On (Ein)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Ausgangseinstellungen wenn *portNumber*=1 (Ausgangsnummer)

OutData Wert	15	14	13	12	11	10	9	8
01	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
02	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)
03	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
08	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)
09	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)
10	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)
11	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
99	Off (Aus)	On (Ein)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Einschränkung

Der Unterschied zwischen OpBCD und Out

Die Befehle Out und **OpBCD** sind einander in der Sprache SPEL sehr ähnlich. Es gibt jedoch einen Hauptunterschied zwischen den beiden. Dieser Unterschied wird im Folgenden aufgezeigt.

- Der OpBCD Befehl verwendet das BCD-Format zur Spezifizierung eines 8-Bit-Wertes der verwendet wird, um die Ausgänge ein- oder auszuschalten. Da das BCD-Format den Gebrauch der Werte &HA, &HB, &HC, &HD, &HE oder &HF ausschließt, können nicht alle Kombinationen für die Einstellung der 8 Ausgänge erfüllt werden.
- Der **Out** Befehl funktioniert ähnlich dem OpBCD Befehl, mit der Ausnahme, dass der Out Befehl es gestattet, den Bereich der 8-Bit-Werte, der für das Ein- oder Ausschalten der Ausgänge verwendet werden darf, zwischen 0 und 255 anzusiedeln (gegenüber 0 bis 99 für OpBCD). Dies gestattet es, alle möglichen Kombinationen für die 8 Ausgangsgruppen den Anwender-Spezifikationen folgend zu initialisieren.

Der Unterschied zwischen Out und Out $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischen den Befehlen Out und **Out $** versteht. Dieser Unterschied wird im Folgenden aufgezeigt.

Der **Out $** -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf den Hardware E/A.

Der **Out**-Befehl arbeitet mit den Hardware-Ausgangsports, die sich auf der Rückseite des Steuergerätes befinden. Diese Hardwareports sind einzelne Ausgänge, die mit Geräten außerhalb der Steuerung kommunizieren.

Verwandte Befehle

In, In $, InBCD, Off, Off $, On, On $, Op, Oport, Out $, Sw, Sw $, Wait

Out Beispiel

Das Beispiel unten zeigt, wie der Haupttask einen Hintergrundtask namens *iotask* startet. *iotask* ist ein einfacher Task, um abwechselnd die Ausgänge 0-3 ein- und dann wieder auszuschalten. Der Out Befehl macht dies möglich, indem nur 1 Befehl verwendet wird, anstatt jeden Ausgang einzeln ein- und auszuschalten.

```
Function main

  Xqt iotask
  Do
    Go P1
    Go P2
  Loop
Fend

Function iotask

  Do
    Out 0, &H0F
    Out 0, &H00
    Wait 10
  Loop
Fend

Weiter einfache Beispiele vom Online-Fenster:

> Out 1,6  'Schaltet die Ausgänge 9 & 10 ein.
> Out 2,1  'Schaltet den Ausgang 8 ein.
> Out 3,91 'Schaltet die Ausgänge 24, 25, 27, 28 und 30 ein
```

Out Funktion

F

Gibt den Status eines Ausgangsbytes aus.

Syntax

Out(*portNumber***)**

Parameter

portNumber	Integer-Ausdruck zwischen 0 und 63, der jeweils eine der 64 Ausgangsgruppen repräsentiert (jede Gruppe enthält 8 Ausgänge), welche die Standard- und die erweiterten Hardwareausgänge einschließen. Die *portNumber*-Auswahl entspricht den folgenden Ausgängen:

Rückgabewerte

Den Ausgangsstatus als 8-Bit-Wert für den spezifizierten Port.

Verwandte Befehle

Out Anweisung

Beispiel einer Out Funktion

```
Print Out(0)
```

Out $ Anweisung

> S

Stellt gleichzeitig 8 S/W Merker ein.

Syntax

Out $ *portNumber, outData*

Parameter

portNumber Integer-Ausdruck zwischen 0 und -63, der jeweils eine der 64 Ausgangsgruppen repräsentiert (jede Gruppe enthält 8 Ausgänge), die die 512 Merker-Ausgänge bilden. Die *portNumber*-Auswahl entspricht den folgenden Ausgängen:

Portnummer	Ausgänge
0	0-7
1	8-15
.	.
63	496-511

outData Integer zwischen 0 und 255, der das Ausgangsmuster für die mit der Anweisung *portNumber* ausgewählte Ausgangsgruppe repräsentiert. Wenn sie in hexadezimaler Form dargestellt wird, wird der Bereich von &H0 bis &HFF abgedeckt. Die niedrigere Ziffer repräsentiert die niederwertigen Stellen (oder die ersten 4 Ausgänge) und die obere Ziffer repräsentiert die höherwertigen Stellen (oder die zweiten 4 Ausgänge).

Beschreibung

Out $ setzt unter Nutzung der vom Anwender angegebenen Kombination aus *portNumber* und *outData* Werten gleichzeitig 8 S/W Merker. Die 512 Merker werden in 64 Gruppen mit jeweils 8 Eingängen unterteilt. Der *portNumber*-Parameter für den Out $-Befehl definiert, welche 8er-Ausgangsgruppe verwendet werden soll, wobei *portNumber* = 0 die Ausgänge 0-7 bedeutet , *portNumber* = 1 bedeutet die Ausgänge 8-15, etc.

Sobald eine *Portnummer* ausgewählt wurde (d.h. es wurde eine Gruppe von 8 Ausgängen ausgewählt), muss ein spezifisches Ausgangsmuster definiert werden. Dafür wird der Parameter *outData* verwendet. Der *outData* Parameter kann einen Wert zwischen 0 und -255 haben und kann in hexadezimalem oder Integer Format repräsentiert werden. (D.h. &H0-&HFF oder 0-255)
Die Tabelle unten zeigt einige der möglichen E/A Kombinationen und ihre entsprechenden *outData*-Werte, davon ausgehend, dass die *portNumber* 0 oder entsprechend 1 ist.

Ausgangseinstellungen wenn *portNumber*=0 (Ausgangsnummer)

OutData Wert	7	6	5	4	3	2	1	0
01	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
02	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)
03	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
08	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)
09	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)
10	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)
11	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
99	Off (Aus)	On (Ein)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Ausgangseinstellungen wenn *portNumber*=1 (Ausgangsnummer)

OutData Wert	15	14	13	12	11	10	9	8
01	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
02	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)
03	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
08	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)
09	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	On (Ein)
10	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	Off (Aus)
11	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)
99	Off (Aus)	On (Ein)	On (Ein)	Off (Aus)	Off (Aus)	Off (Aus)	On (Ein)	On (Ein)
255	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)	On (Ein)

Einschränkung

Der Unterschied zwischen Out und Out $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischen den Befehlen Out und **Out $** versteht. Dieser Unterschied wird im Folgenden aufgezeigt:

Der **Out $** Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf die Hardware E/As.

- Der Out-Befehl arbeitet mit den Hardware-Ausgangsports, die sich auf der Rückseite des Steuergerätes befinden. Diese Hardwareports sind einzelne Ausgänge, die mit Geräten außerhalb der Steuerung kommunizieren.

Verwandte Befehle

In, In$, InBCD, Off, Off$, On, On$, OpBCD, Oport, Out, Sw, Sw$, Wait

Out $ Beispiel

Das Beispiel unten zeigt, wie der Haupttask einen Hintergrundtask namens *iotask* startet. *iotask* ist ein einfacher Task, um abwechselnd S/W E/A Nrn. 0-3 ein- und dann wieder auszuschalten. Der Out $ Befehl macht dies möglich, indem nur 1 Befehl verwendet wird, anstatt jeden S/W Merker einzeln mit On $ und Off $ zu setzen.

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
Fend

Function iotask

  Out $ 0,&H0F  Out $ 0,&H0  Wait 0  GoTo 0Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> Out 1,6   'Schaltet die Ausgänge 9 & 10 ein.
> Out 2,1   'Schaltet den Ausgang 8 ein.
> Out 3,91  'Schaltet die Ausgänge 24, 25, 27, 28 und 30 ein
```

OutW Anweisung

Setzt 16 Ausgänge gleichzeitig.

Syntax

OutW *wordPortNum, outputData*

Parameter

wordPortNum	Spezifiziert WordPort-Nummern (Integer zwischen 0 und 31, innerhalb des installierten E/A-Bereichs) unter Verwendung eines Ausdrucks oder eines numerischen Wertes.
outputData	Spezifiziert Ausgangsdaten (Integers von 0 bis 65535) unter Verwendung eines Ausdrucks oder eines numerischen Wertes.

Beschreibung

Ändert den aktuellen Status der E/A -Ausgangsportgruppe, spezifiziert durch die WordPort-Nummer, in die angegebenen Ausgangsdaten.

Verwandte Befehle

In, InW, Out

OutW Beispiel

```
OutW 0, 25
```

OutW Funktion

F

Gibt den Status eines Ausgangswortes (2 Bytes) aus.

Syntax

OutW(*wordPortNum***)**

Parameter

wordPortNum	Spezifiziert WordPort-Nummern (Integer zwischen 0 und 31, innerhalb des installierten E/A Bereichs) unter Verwendung eines Ausdrucks oder eines numerischen Wertes.

Rückgabewerte

Der Ausgangsstatus 16-Bit-Wert für den spezifizierten Port.

Verwandte Befehle

OutW Anweisung

Beispiel einer OutW Funktion

```
OutW 0, &H1010
```

PAgl Funktion

F

Gibt den Achswert eines bestimmten Punktes aus.

Syntax

PAgl (*point*, *jointNumber)*

Parameter

point-{}-	Punktausdruck.
jointNumber	Definiert die Achsnummer (ganze Zahl von 1 bis 4), repräsentiert durch einen Ausdruck oder einen numerischen Wert.

Rückgabewerte

Gibt die berechnete Achsposition aus (Real-Wert, deg für eine Rotationsachse, mm für eine lineare Achse).

Verwandte Befehle

Ag, CU, CX, CY, CZ, PPls

Beispiel einer PAgl Funktion

```
Real joint1

joint1 = PAgl(P10, 1)
```

Pallet Anweisung

> S

Definiert Paletten und zeigt sie an.

Syntax

Pallet [*palletNumber, Pi, Pj, Pk* [,*Pm*], *columns, rows*]

Parameter

palletNumber	Palettennummer, repräsentiert durch einen Integer zwischen 0 und 15.
Pi, Pj, Pk	Punktvariablen, welche die Standard-3-Punkte-Palette definieren.
Pm	Optional. Punktvariable, die mit Pi, Pj und Pk verwendet wird, um eine 4-Punkte-Palette zu definieren.
columns	Integer Ausdruck, der die Anzahl von Punkten auf der Pi-bis-Pj-Seite der Palette repräsentiert. Der Bereich liegt zwischen 1 und 32767.
rows	Integer Ausdruck, der die Anzahl von Punkten auf der Pi-bis-Pk-Seite der Palette repräsentiert. Der Bereich liegt zwischen 1 und 32767.

Rückgabewerte

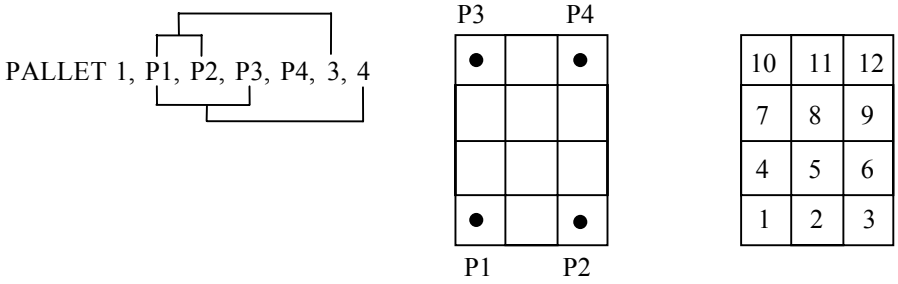
Wenn die Parameter weggelassen werden, werden alle definierten Paletten angezeigt.

Beschreibung

Definiert eine Palette, indem dem Roboter mindestens die Punkte Pi, Pj und Pk eingeteacht werden und indem die Anzahl von Punkten von Pi bis Pj und von Pi bis Pk angegeben wird.

Wenn die Palette eine gleichmäßig-rechteckige Form hat, müssen nur 3 der 4 Eckpunkte definiert werden. In den meisten Fällen ist es jedoch besser, 4 Eckpunkte zu verwenden, um eine Palette zu definieren.

Um eine Palette zu definieren, teachen Sie dem Roboter zunächst entweder 3 oder 4 Eckpunkte ein und definieren Sie dann die Palette wie folgt:
Eine mit 4 Punkten definierte Palette: P1, P2, P3 und P4 wird unten gezeigt. Es gibt 3 Positionen von P1-P2 und 4 Positionen von P1-P3. Dies ergibt eine Palette mit einer Gesamtzahl von 12 Positionen. Die Syntax für die Definition dieser Palette lautet wie folgt:



Punkten, die Teile einer Palette repräsentieren, werden automatisch Teilnummern zugeordnet, die in diesem Beispiel bei P1 beginnen. Diese Teilnummern werden auch von der Pallet Funktion benötigt.

Hinweise

Die Maximale Palettengröße:

Die Gesamtanzahl von Punkten, die durch eine spezifische Palette definiert ist, muss weniger als 32767 betragen.

Nicht korrekte Palettenform-Definitionen:

Beachten Sie bitte, dass eine falsche Punktreihenfolge oder eine falsche Anzahl von Teilen zwischen den Punkten eine nicht korrekte Palettenform-Definition zur Folge hat.

Definition der Palettenebene:

Die Palettenebene wird durch Z-Achsen-Koordinatenwerte der 3 Palletteneckpunkte definiert. Daher können auch vertikale Paletten definiert werden.

Palettendefinition für eine Palette mit einer einzigen Reihe:

Eine Palette mit nur einer einzigen Reihe kann mit einer 3-Punkte Pallet-Anweisung oder einem - Befehl definiert werden. Teachen Sie einfach einen Punkt an jedem Ende ein und definieren Sie dann wie folgt: Spezifizieren Sie 1 als Anzahl der Teile zwischen demselben Punkt.

```
> Palette 2, P20, P21, P20, 5, 1        'Definiert eine 5x1 Palette
```

Verwandte Befehle

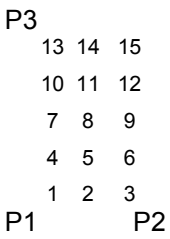
Pallet Funktion

Beispiel einer Pallet Anweisung

Der folgende Befehl vom Online-Fenster erstellt die Palette, die durch die Punkte P1, P2 und P3 definiert ist und teilt die Palettenebene in 15 gleich verteilte Palettenpunkt-Positionen, bei denen Palettenpunkt Nr. 1, Palettenpunkt Nr. 2 und Palettenpunkt Nr. 3 entlang der Seite P1-bis-P2 angeordnet sind.

```
> pallet 1, P1, P2, P3, 3, 5
> jump pallet(1, 2)          'Springt zur Position auf der Palette
```

Die resultierende Palette sieht aus wie folgt:



Pallet Funktion

F

Spezifiziert eine Position in einer vorher definierten Palette.

Syntax

Pallet (*palletNumber*, *palletPosition* **)**
Pallet (*palletNumber*, *column, row* **)**

Parameter

palletNumber	Palettennummer, repräsentiert durch einen Integer Ausdruck zwischen 0 und 15.
PalletPosition	Die Palettenposition wird durch einen Integer zwischen 1 und 32767 repräsentiert.
column	Die Palettenspalte wird durch einen Integer Ausdruck zwischen 1 und 32767 repräsentiert.
row	Die Palettenzeile wird durch einen Integer Ausdruck zwischen 1 und 32767 repräsentiert.

Beschreibung

Pallet gibt eine Position in einer Palette aus, die zuvor durch die Pallet Anweisung definiert wurde. Verwenden Sie die Funktion zusammen mit den Befehlen Go oder Jump, um den Arm zu der spezifizierten Palettenposition zu bewegen.

Die Palettenpositionsnummer kann arithmetisch oder einfach unter Verwendung eines Integers definiert werden.

Verwandte Befehle

Pallet Anweisung

Beispiel einer Pallet Funktion

Das folgende Programm transferiert Teile von Palette 1 nach Palette 2.

```
Function main
  Integer index
  Pallet 1 P1, P2, P3, 3, 5      'Definiert Pallete 1
  Pallet 2 P12, P13, P11, 5, 3   'Definiert Pallete 2
  For index = 1 To 15
    Jump Pallet(1, index)     'Bewegt den Arm zum Punkt-Index auf
Palette 1
    On 1     'Hält das Werkstück
    Wait 0.5
    Jump Pallet(2, index)     'Bewegt den Arm zum Punkt-Index auf
Palette 2
    Off 1     'Gibt das Werkstück frei
    Wait 0.5
  Next I
Fend
```

ParseStr Anweisung / Funktion

F S

Analysiert eine Zeichenkette syntaktisch und gibt eine Textelement-Matrix aus.

Syntax

ParseStr *inputString$, tokens$(), delimiters$*
numTokens = **ParseStr(***inputString$, tokens$(), delimiters$***)**

Parameter

inputString$	Zeichenkettenausdruck, der syntaktisch analysiert (geparst) werden soll.
tokens$()	Ausgangsmatrix von Zeichenketten, welche die Textelemente enthalten.
delimiters$	Zeichenkettenausdruck, der einen oder mehrere Textelement-Begrenzungen enthält.

Rückgabewerte

Wird ParseStr als Funktion verwendet, wird die Anzahl der geparsten Textelemente ausgegeben.

Verwandte Befehle

Redim, String

Beispiel einer ParseStr Anweisung

```
String toks$(0)
Integer i

ParseStr "1 2 3 4", toks$(), " "

For i = 0 To UBound(toks)
   Print "token ", i, " = ", toks$(i)
Next i
```

Pass Anweisung

> S

Führt gleichzeitig vier PTP-Bewegungen aus und bewegt den Manipulatorarm dabei an festgelegten Punkten vorbei (Verschleifen von Hilfspunkten, ohne diese exakt zu überfahren).

Syntax

Pass *point* [, {**On | Off**} *bitNumber* [*, point ...*]

Parameter

point	**P***number* oder **P**(*expr*) oder Punktlabel.
bitNumber	Der ein- oder auszuschaltende Ausgang. Integer zwischen 0 und 511 oder Ausgangslabel.

Beschreibung

Der **Pass**–Befehl bewegt den Roboterarm in die Nähe, aber nicht direkt durch die spezifizierte Punktserie.

Um eine Punktserie zu spezifizieren, verwenden Sie Punkte (P0,P1, ...) und trennen Sie diese durch Kommata.

Um Ausgänge während der Bewegung ein- oder auszuschalten, fügen Sie einen durch Kommata abgetrennten On- oder Off-Befehl zwischen die Punkte ein. On oder Off wird ausgeführt, bevor der Roboter den Punkt erreicht, der genau vor On oder Off kommt.

Wenn auf **Pass** direkt ein weiterer **Pass-**Befehl folgt, wechselt die Steuerung zum folgenden **Pass-**Befehl, ohne dass der Roboter am letzen definierten Punkt des vorangegangenen **Pass-**Befehls anhält.

Wenn direkt auf **Pass** ein Bewegungsbefehl folgt, der nicht **Pass** lautet, hält der Roboter am letzen definierten Punkt des vorangegangenen **Pass-**Befehls an. Eine **Fine**-Positionierung wird jedoch nicht durchgeführt.

Wenn direkt auf **Pass** ein Befehl, eine Anweisung oder eine Funktion folgt, die kein Bewegungsbefehl ist, kann der direkt darauf folgende Befehl, die Anweisung oder die Funktion ausgeführt werden, bevor der Roboter den letzen Punkt des vorangegangenen Pass-Befehls erreicht hat.

Wenn eine Fine-Positionierung an der Zielposition gewünscht wird, lassen Sie auf den Pass- einen Go-Befehl folgen und spezifizieren Sie die Zielposition wie im folgenden Beispiel gezeigt.

```
Pass P5; Go P5; On 1; Move P10
```

Je größer die Werte für Beschleunigung und Verzögerung sind, desto näher fährt der Arm an den angegebenen Punkt. Der **Pass-**Befehl kann auch so verwendet werden, dass der Roboterarm Hindernisse umgeht.

Einschränkung

SPEL$^+$ unterstützt keine CP-Spezifikation (Continuous Point Spezifikation)

In der Version der SPEL-Sprache, die für SRC 3xx Controller verwendet wurde, war es erlaubt, einen fortlaufenden Punktesatz zu spezifizieren. Zum Beispiel:

```
Pass P1-P10
```

In der SPEL$^+$-Sprache bedeutet die obige Anweisung jedoch, dass eine Bewegung zu einem einzigen Punkt ausgeführt wird (P1 minus P10).
Um einen fortlaufenden Punktesatz zu definieren, müssen Sie alle Punkte spezifizieren.

Verwandte Befehle

Accel, Go, Jump, Speed

Pass Beispiel

Das Beispiel zeigt die Handhabung des Roboterarms mit Hilfe des Pass-Befehls:

```
Function main
  Jump P1
  Pass P2   'Bewegt den Arm zu P2 und  führt
            'den nächsten Befehl durch, bevor P2 erreicht wird.
  On 2
  Pass P3
  Pass P4
  Off 0
  Pass P5
Fend
```

Pause Anweisung

S

Vorübergehender Stopp der Ausführung aller Programme, für welche die Pause-Anweisung aktiviert ist.

Syntax

Pause

Beschreibung

Wenn die **Pause**-Anweisung ausgeführt wird, wird die Programmausführung für all jene Tasks unterbrochen, für die *Pause* aktiviert ist. Ebenso werden alle Tasks unterbrochen, die gerade einen Bewegungsbefehl ausführen, selbst wenn *Pause* für diese Tasks nicht aktiviert ist.

Einschränkung

QP und die Auswirkungen auf Pause:

Der QP-Befehl wird verwendet, um den Arm direkt bei *Pause* zu stoppen, oder um die aktuelle Bewegung abzuschließen und das Programm dann zu pausieren. Bitte lesen Sie die Hilfe-Funktion des QP-Befehls für weitere Informationen.

Verwandte Befehle

Cont, Resume

Beispiel einer Pause Anweisung

Das Beispiel unten zeigt die Verwendung des **Pause**-Befehls zur vorübergehenden Aussetzung der Programmausführung. Der Task führt Programmanweisungen aus, bis er die Zeile erreicht, die einen Pause-Befehl enthält. An diesem Punkt wird der Task ausgesetzt. Der Anwender kann dann auf den Button *Continue* im Run Window klicken, um mit der Ausführung fortzufahren.

```
Function main

  Xqt monitor, NoPause  ' Dieser Task wird nicht angehalten
  Go P1
  On 1
  Jump P2
  Off 1
  Pause    'Unterbricht die Programmausführung
  Go P40
  Jump P50
Fend
```

PauseOn Funktion

F

Gibt den aktuellen *Pause*-Status aus.

Syntax
PauseOn

Rückgabewerte
TRUE, wenn sich der Controller im Pause-Status befindet, sonst FALSE.

Beschreibung
PauseOn gibt TRUE aus, wenn sich der Controller im Pause-Status befindet.

PauseOn entspricht `((Stat(0) And &H20000)<>0)`.

Verwandte Befehle
EStopOn, SafetyOn, Stat

Beispiel einer PauseOn Funktion

```
If PauseOn Then
    Print "Pause condition is On"
EndIf
```

PDef Funktion

F

Gibt den Definitionsstatus eines spezifizierten Punktes aus.

Syntax
PDef (*Point*)

Parameter

point	Integer Ausdruck oder **P***number* oder **P**(*expr*) oder Punktlabel.

Rückgabewerte
TRUE, wenn der Punkt definiert ist, sonst FALSE.

Verwandte Befehle
Here Anweisung, PDel

Beispiel einer PDef Funktion

```
If Not PDef(1) Then
    Here P1
Endif
```

PDel

>

Löscht spezifizierte Positionsdaten.

Syntax

PDel *firstPointNum ,* [*lastPointNum*]

Parameter

firstPointNum	Die erste Punktnummer in einer zu löschenden Punktsequenz. *firstPointNum* muss ein Integer sein.
lastPointNum	Die letzte Punktnummer in einer zu löschenden Punktsequenz. *lastPointNum* muss ein Integer sein.

Beschreibung

Löscht spezifizierte Postitionsdaten aus dem Controller-Punktspeicher für den aktuellen Roboter. Löscht alle Positionsdaten angefangen bei *firstPointNum* bis *lastPointNum* einschließlich. Um zu verhindern, dass Fehler Nr. 2 auftritt, muss *firstPointNum* kleiner sein als *lastPointNum*.

PDel Beispiel

```
> p1=10,300,-10,0/L
> p2=0,300,-40,0
> p10=-50,350,0,0
> pdel 1,2       'Löscht die Punkte 1 und 2
> plist
P10  =   -50.000,  350.000,    0.000,    0.000 /R /0
> pdel 50        'Löscht Punkt 50
> pdel 100,200   'Löscht die Punkte 100 bis 200
>
```

PList

>

Zeigt Punktdaten an.

Syntax

Plist *	(Zeigt die Koordinatendaten der aktuellen Armposition an.)
Plist *StartPoint* , [*EndPoint*]	(Listet die spezifizierten Anfangs- und Endpunkte auf.)

Parameter

StartPoint	Der Startpunkt-Index. Der Indexbereich beträgt 0 bis (“number of points” - 1).
EndPoint	Der Endpunkt-Index. Der Indexbereich beträgt 0 bis (“number of points” - 1).

Rückgabewerte

Punktdaten.

Beschreibung

Plist zeigt alle Punktdaten an, die durch den Bereich Start- und Endpunkt spezifiziert werden.

Wenn keine Punktdaten im spezifizierten Punktbereich vorhanden sind, oder ein Startpunkt-Index größer als der Endpunkt-Index spezifiziert ist, werden keine Daten angezeigt.

PList Beispiel

Dieser Plist-Befehl im Online-Fenster zeigt die Punktdaten der aktuellen Armposition an.

```
> plist *
     300   150
       0     0
>
```

Dieses Beispiel zeigt die spezifizierten Punktdaten an.

```
> plist 2
p2=-160,400,0,0
>
```

Dieses Beispiel zeigt die Punktdaten innerhalb des Bereiches von 10 und 20 an. In diesem Beispiel befinden sich in diesem Bereich lediglich drei Punkte.

```
> plist 10 - 20
p10=160, 100, 0, 0
p12= 55, 340, 0, 0
p20= 50, 200, -100, -100
>
```

Dieser Plist-Befehl zeigt die Punktedaten innerhalb eines Bereichs von 0 und dem spezifizierten Endpunkt an.

```
> plist - 20
p2=-160,400,0,0
p10=160, 100, 0, 0
p12= 55, 340, 0, 0
p20= 50, 200, -100, -100
>
```

Plist in einem festen Format.

```
> Plist /W
P000=  450.000,   400.000,   0.000,   0.000
P001=    1.325,   330.170, -58.200,  -8.000  /L
P011= -200.000,   400.000,-100.000, 150.000/1/R
```

PLabel$ Funktion

Gibt das mit einer Punktnummer verbundene Punktlabel aus.

Syntax

PLabel$(*pointNumber***)**

Parameter

pointNumber Ein Integer Ausdruck, der eine Punktnummer enthält, die in der aktuellen Punktedatei verwendet wird.

Verwandte Befehle

PDef Function, PNumber Function

Beispiel einer PLabel$ Funktion

```
Print PLabel$(1)
```

PLocal Anweisung

> S

Stellt das lokale Attribut für einen Punkt ein.

Syntax

PLocal(*point***) =** *localNumber*

Parameter

point	**P***number* oder **P**(*expr*) oder Punktlabel.
localNumber	Ein Integer Ausdruck, der die neue Local-Nummer repräsentiert. Der Bereich liegt zwischen 0 und 15.

Verwandte Befehle

PLocal Funktion

Beispiel einer PLocal Anweisung

```
PLocal(pick) = 1
```

PLocal Funktion

F

Gibt die lokale Nummer für einen spezifizierten Punkt aus.

Syntax
PLocal(*point***)**
Parameter
point **P***number* oder **P**(*expr*) oder Punktlabel.

Rückgabewerte
Local-Nummer für einen spezifizierten Punkt.

Verwandte Befehle
PLocal Anweisung

Beispiel einer PLocal Funktion

```
Integer localNum

localNum = PLocal(pick)
```

Pls Funktion

F

Gibt den aktuellen Encoder-Pulswert für jede Achse an der aktuellen Position aus.

Syntax

Pls*(jointNumber)*

Parameter

jointNumber Die spezifische Achse, für die der aktuelle Encoder-Pulswert ermittelt werden soll.

1 - (XM X Achse (Linearachse), SCARA 1. Achse (Rotationsachse))

2 - (XM Y Achse (Linearachse), SCARA 2. Achse (Rotationsachse))

3 - Z Achse (3. Achse (Linearachse))

4 - U Achse (4. Achse (Rotationsachse)

Rückgabewerte

Gibt einen Nummernwert aus, der den aktuellen Encoder-Pulswert für die durch *jointNumber* spezifizierte Achse ausgibt.

Beschreibung

Pls wird verwendet, um die aktuelle Encoderposition (Pulswert) jeder Achse zu lesen. Diese Werte können gespeichert und später mit dem Pulse-Befehl verwendet werden.

Verwandte Befehle

CU, CX, CY, CZ, Pulse

Beispiel einer Pls Funktion

Im Folgenden wird ein einfaches Beispiel gezeigt, wie man die Pulse-Werte für jede Achse erhält und sie druckt.

```
Function plstest
  Real t1, t2, z, u
  t1 = pls(1)
  t2 = pls(2)
  z = pls(3)
  u = pls(4)
  Print "T1 joint current Pulse Value: ", t1
  Print "T2 joint current Pulse Value: ", t2
  Print "Z joint current Pulse Value: ", z
  Print "U joint current Pulse Value: ", u
Fend
```

PNumber Funktion

Gibt die mit einem Punktlabel verbundene Punktnummer aus.

Syntax

PNumber(*pointLabel***)**

Parameter

pointLabel Ein Punktlabel, das in der aktuellen Punktdatei verwendet wird oder ein Zeichenkettenausdruck, der ein Punktlabel enthält.

Verwandte Befehle

PDef Funktion, PLabel$ Funktion

Beispiel einer PNumber Funktion

```
Integer pNum
String pointName$

pNum = PNumber(pick)

pNum = PNumber("pick")

pointName$ = "place"
pNum = PNumber(pointName$)
```

Punkt-Zuweisung

> S

Definiert einen Roboterpunkt, indem er einem Punktausdruck zugeordnet wird.

Syntax
point = pointExpr

Parameter

point Ein Roboterpunkt spezifiziert wie folgt:
P*number*
P(*expr*)
pointLabel

pointExpr Punktausdruck.

Beschreibung

Definiert einen Roboterpunkt, indem er einem anderen Punkt oder Punktausdruck gleichgesetzt wird.

Verwandte Befehle
Local, Pallet, PDef, PDel, Plist

Punkt-Zuweisung Beispiel

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt:

Zuordnung von Koordinaten zu P1:

```
> P1 = 300,200,-50,100
```

Spezifizierung der Linksarmstellung:

```
> P2 = -400,200,-80,100/L
```

Addieren Sie 20 zur X-Koordinate von P2 und definieren Sie den resultierenden Punkt als P3:

```
> P3 = P2 +X(20)
> plist 3
P3=-380,200,-80,100/L
```

Subtrahieren Sie 50 von der P2- Y-Koordinate, ersetzen Sie die Z-Koordinate durch -30, und definieren Sie den resultierenden Punkt P4 als Stellung des rechten Arms:

```
>P4=P2 -Y(50) :Z(-30) /R
```

Addieren Sie 90 zur U-Koordinate der Palette(n) (3, 5), und definieren Sie den resultierenden Punkt als P6:

```
> P5 = P*
> P6 = pallet(3,5) +U(90)
```

Punktausdruck

> S

Spezifiziert einen Roboterpunkt für die Zuordnung oder für einen Bewegungsbefehl.

Syntax

point [{ + | - } point] [orientation] [localNumber] [relativeOffsets] [absoluteCoords]

Parameter

point	Spezifizierung des Basis-Punktes. Dabei kann es sich um eine der folgenden Optionen handeln: **P***number* **P**(*expr*) **P*** **Here** **Pallet**(*palletNumber*, *palletIndex*) *pointLabel* *XcoordExpr*, *YcoordExpr*, *ZcoordExpr*, *UcoordExpr* **XY**(*XcoordExpr*, *YcoordExpr*, *ZcoordExpr*, *UcoordExpr*) **JA**(*j1angle*, *j2angle*, *j3angle*, *j4angle*) **Pulse**(*j1pulses*, *j2pulses*, *j3pulses*, *j4pulses*)
orientation	Für SCARA Roboter optional. Spezifizieren Sie /**L** oder /**R** für *lefty* oder *righty* - Ausrichtung.
localNumber	Optional. Local-Nummer von 1 bis 15 mit einem vorangestellten Schrägstrich (/1 bis /15).
relativeOffsets	Optional. Eine oder mehrere relative Koordinaten. {**+** | **-**} {**X** | **Y** | **Z** | **U**} **(***expr***)**
absoluteCoords	Optional. Eine oder mehrere absolute Koordinaten. **:** {**X** | **Y** | **Z** | **U**} **(***expr***)**

Beschreibung

Punktausdrücke werden in Punktzuordnungs-Anweisungen und Bewegungsbefehlen verwendet. Man kann Punkte addieren und subtrahieren, wenn keine direkten Koordinaten verwendet werden. Zum Beispiel:

```
Go P1 + P2
P1 = P2 + XY(100, 100, 0, 0)
```

Verwendung des relativen Versatzes

Sie können eine oder mehrere Koordinaten relativ zum Basis-Punkt versetzen. Beispielsweise bewegt die folgende Anweisung den Roboter aus der aktuellen Position 20 mm die positive X-Achse entlang:

```
Go P* +X(20)
```

Wenn Sie die gleiche Anweisung erneut ausführen, bewegt sich der Roboter erneut um 20mm an der X-Achse entlang, da dies eine relative Bewegung ist.

Verwendung absoluter Koordinaten

Sie können eine oder mehrere Koordinaten des Basis-Punktes durch Verwendung absoluter Koordinaten ändern. Die folgende Anweisung bewegt den Roboter zur 20mm-Position auf der X-Achse:

```
Go P* :X(20)
```

Wenn Sie dieselbe Anweisung erneut ausführen, bewegt sich der Roboter nicht, da er sich auf Grund der vorangegangenen Bewegung bereits in der absoluten Position für X befindet.

Relativer Versatz und absolute Koordinaten machen es einfach, einen Punkt vorübergehend zu modifizieren. Beispielsweise bewegt dieser Code den Roboter schnell an die Position 10 mm über den Pick-Punkt. Dabei wird ein relativer Versatz für Z von 10 mm verwendet. Anschließend bewegt sich der Roboter dann langsam zum Pick-Punkt selbst.

```
Speed fast
Jump pick +Z(10)
Speed slow
Go pick
```

Dieser Code bewegt den Roboter von der aktuellen Position direkt aufwärts, indem ein absoluter Wert von 0 (Null) für die Z-Achse spezifiziert wird:

```
LimZ 0
Jump P* :Z(0)
```

Verwendung von lokalen Koordinatensystemen (locals)

Sie können mit einem Schrägstrich oder einem @-Zeichen eine Local-Nummer spezifizieren. Jedes dieser Zeichen hat eine separate Funktion.

Verwenden Sie den Schrägstrich, um die Koordinaten einer Local-Nummer zu kennzeichenen

Beispielsweise bedeutet das Hinzufügen von /1 in der folgenden Anweisung, dass P1 sich am Ort 0,0,0,0 in Local 1 befindet.

```
P1 = 0, 0, 0, 0 /1
```

Das @-Zeichen wird verwendet, um Koordinaten in Local-Koordinaten zu übertragen. Dieses Beispiel zeigt, wie ein Punkt in einer Local-Nummer eingeteacht wird:

```
P1 = P* @1
```

Die aktuelle Position wird in ihre Position in Local 1 übertragen.

Verwandte Befehle

Go, Local, Pallet, Pdel, Plist

Beispiel einer Point Expression (Punktausdruck)

Hier sind einige Beispiele der Verwendung von Punktausdrücken in Zuordnungsanweisungen und Bewegungsbefehlen:

```
P1 = 300,200,-50,100
P2 = P1 /R
P3 = pick /1
P4 = P5 + P6
P(i) = 100, 200, CZ(P100), 0
Go P1 -X(20) :Z(-20) /R
Go Pallet(1, 1) -Y(25.5)
Move pick /R
Jump Here :Z(0)
Go P* :Z(-25.5)
Go JA(25, 0, -20, 180)
pick = 100, 100, -50, 0
```

POrient Anweisung

> S

Stellt die Ausrichtung für einen Punkt ein.

Syntax

POrient(*point***) =** *orientation*

Parameter

point	**P***number* oder **P**(*expr*) oder Punktlabel.
orientation	Integer Ausdruck, der den neuen Ausrichtungswert darstellt. 0 Vorgabeeinstellung 1 RIGHTY 3 LEFTY

Verwandte Befehle

POrient Funktion

Beispiel einer POrient Anweisung

```
POrient(pick) = RIGHTY
POrient(1) = LEFTY
```

POrient Funktion

F

Gibt die Ausrichtung eines Punktes aus.

Syntax

POrient(*point***)**

Parameter

point	Integer Ausdruck, der eine Punktnummer oder ein Punktlabel repräsentiert.

Rückgabewerte

0 Vorgabeeinstellung
1 RIGHTY
3 LEFTY

Verwandte Befehle

POrient Anweisung

Beispiel einer POrient Funktion

```
Print POrient(pick)
Print POrient(1)
Print POrient(P1)
```

PosFound Funktion

F

Gibt den Status der Find-Operation aus.

Syntax

PosFound

Rückgabewerte

TRUE, wenn die Position während der Bewegung gefunden wurde. FALSE, wenn die Position nicht gefunden wurde.

Verwandte Befehle

Find

Beispiel einer PosFound Funktion

```
Find Sw(5) = ON
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Error: Cannot find the sensor signal."
EndIf
```

Power Anweisung

> S

In älteren Fassungen Lp genannt.

Schaltet den Power-Modus auf High oder Low und zeigt den aktuellen Status an.

Power Syntax

Power [{ **High | Low** }]

Parameter

High | Low Die Einstellung kann High oder Low sein. Die Vorgabeeinstellung ist Low.

Rückgabewerte

Zeigt den aktuellen Power Status an, wenn der Parameter weggelassen wird.

Beschreibung

Die Anweisung schaltet den Power-Modus auf High oder Low. Außerdem wird der aktuelle Modusstatus angezeigt.

On – Wenn Power auf Low eingestellt ist, ist der Low Power-Modus eingeschaltet. Das bedeutet, dass der Roboter langsam verfährt (unter 250 mm/sec) und die Servo-Steifigkeit auf niedrig eingestellt wird, damit die Servoachse freigeschaltet wird, wenn der Roboter gegen ein Objekt stößt. Dies ist die normale Betriebsart zum Teachen von Punkten.

On – Wenn Power auf High eingestellt ist, ist der Low Power-Modus ausgeschaltet. Das bedeutet, dass der Roboter mit voller Geschwindigkeit und mit voller Servo-Steifigkeit verfahren kann. Dies ist die normale Betriebsart für den Ablauf tatsächlicher Anwendungen.

Die folgenden Operationen bewirken ein Umschalten in den Low-Power-Modus. In diesem Fall werden die Einstellungen für Geschwindigkeit und Beschleunigung auf niedrige Werte begrenzt.

Conditions to Cause Power Low

- Reset Command
- Motor On
- All tasks aborted
- Attend mode ON

→ **Power Low**

Values Limited

Speed
Accel
SpeedS
AccelS

Hinweise

Der Low Power-Modus (Power Low) und seine Auswirkungen auf die maximale Geschwindigkeit (Max Speed):

Im Low Power-Modus ist die Motorkraft begrenzt und die Einstellung der effektiven Bewegungsgeschwindigkeit ist niedriger als der Vorgabewert. Wenn im Low Power-Modus eine höhere Geschwindigkeit (direkt) vom Online-Fenster aus oder in einem Programm spezifiziert wurde, wird die Geschwindigkeit auf den Vorgabewert gesetzt. Wenn eine höhere Bewegungsgeschwindigkeit benötigt wird, setzen Sie Power auf High.

Der High Power-Modus (Power High) und seine Auswirkungen auf die maximale Geschwindigkeit (Max Speed):

Im High Power-Modus können höhere Geschwindigkeiten als der Vorgabewert eingestellt werden.

Verwandte Befehle

Accel, AccelS, Speed, SpeedS, TSpeed, TSpeedS

Power Beispiel

Die folgenden Beispiele werden vom Online-Fenster aus ausgeführt:

```
> Speed 50  'Bestimmt hohe Geschwindigkeit im Low Power-Modus

> Accel 100, 100  'Spezifiziert hohe Beschleunigung

> Jump P1   'Bewegt den Roboter mit niedriger Geschwindigkeit und
niedriger Beschleunigung

> Speed         'Zeigt die aktuellen Geschwindigkeitswerte an
Low Power Mode
   50
   50     50

> Accel         'Zeigt die aktuellen Beschleunigungswerte an
Low Power Mode
   100    100
   100    100
   100    100

> Power High   'Setzt den High Power-Modus

> Jump P2   'Bewegt den Roboter mit hoher Geschwindigkeit
```

Power Funktion

F

Gibt den Power-Status aus.

Syntax

Power

Rückgabewerte

0 = Power Low, 1 = Power High.

Verwandte Befehle

Power Anweisung

Beispiel einer Power Funktion

```
If Power = 0 Then
    Print "Low Power Mode"
EndIf
```

PPls Funktion

F

Gibt die Pulse-Position einer bestimmten Achse für einen angebenen Punkt aus.

Syntax

PPls (*point*, *jointNumber)*

Parameter

point	Punktausdruck.
jointNumber	Definiert die Achsnummer (ganze Zahl von 1 bis 4), unter Verwendung eines Ausdruckes oder eines numerischen Wertes.

Rückgabewerte

Gibt die errechnete Achsposition aus (Long-Wert, in Pulsen).

Verwandte Befehle

Agl, CU, CX, CY, CZ, PAgl

PPls Beispiel

```
Integer pulses1

pulses1 = PPls(P10, 1)
```

Print Anweisung

> S

Gibt Daten zum aktuellen Anzeigefenster aus, einschließlich des Run-Fensters, Operator-Fensters, Online-Fensters und Makro-Fensters.

Syntax

Print [*expression* [*, expression...*] [,]

Parameter

expression	Optional. Eine Zahl oder ein Zeichenkettenausdruck.
, (comma)	Optional. Wenn am Ende der Anweisung ein Komma steht, wird kein CRLF hinzugefügt.

Rückgabewerte

Variablendaten oder die spezifizierte Buchstabenzeichenkette.

Beschreibung

Print zeigt die Variablendaten oder die Buchstabenzeichenkette auf dem Steuergerät an.

Ans Ende der Zeile wird automatisch CRLF (carriage return and line feed / Wagenrücklauf mit Zeilenvorschub) angehängt, es sei denn, am Ende der Anweisung wird ein Komma verwendet.

Verwandte Befehle

Print #, LPrint, EPrint

Beispiel einer Print Anweisung

Das folgende Beispiel extrahiert denWert der U-Achsen-Koordinate von Punkt P100 und verlegt den Koordinatenwert in die Variable *uvar*. Der Wert wird dann auf dem aktuellen Anzeigefenster ausgegeben.

```
Function test
  Real uvar
  uvar = CU(P100)
  Print "The U Axis Coordinate of P100 is ", uvar
Fend
```

Print # Anweisung

> S

Gibt Daten an die angegebene Kommunikationsschnittstelle aus.

Syntax

Print #*PortNumber,* [*Arg1, Arg2, Arg3,...*]

Parameter

PortNumber Nummer der Kommunikationsschnittstelle. Einer der Standard RS232-Ports (1 bis 16). Der Portnummer sollte immer ein Nummern-Zeichen (#) vorangestellt werden. (z.B. #1).

Arg1, Arg2, Arg3... Jedes Argument kann ein Variablenname, ein numerischer Wert oder eine Buchstabenzeichenkette sein.

Beschreibung

Print # gibt den numerischen Wert oder die Buchstabenzeichenkette an die Kommunikationsschnittstelle aus, die durch die Portnummer spezifiziert wird.

Verwandte Befehle

Print

Print # Beispiel

Im Folgenden sehen Sie einige Print# Beispiele:

```
Function printex
  String temp$
  Print #1, "5"     'Sendet das Zeichen "5" an die Serielle
                        Schnittstellt 1
  temp$ = "hello"
  Print #1, temp$
  Print #2, temp$
  Print #1 " Next message for port 1"
  Print #2 " Next message for port 2"
Fend
```

PTran Anweisung

S

Führt eine relative Bewegung einer Achse in Pulsen aus.

Syntax

PTran *joint, pulses*

Parameter

joint	Integer Ausdruck, der die zu bewegende Achse repräsentiert.
pulses	Integer Ausdruck, der die Anzahl der zu verfahrenden Pulse repräsentiert.

Beschreibung

PTran wird verwendet, um eine Achse über eine definierte Anzahl von Pulsen von der aktuellen Position zu bewegen.

Verwandte Befehle

Go, JTran, Jump, Move

Beispiel einer PTran Anweisung

```
PTran 1, 2000
```

Pulse Funktion

F

Gibt einen Roboterpunkt aus, dessen Koordinaten in Pulsen spezifiziert sind.

Syntax

Pulse (*joint1, joint21, joint31, joint4*)

Parameter

joint1, *joint2*, *joint3*, *joint4*	Der Pulswert für jede der vier Achsen. Der Pulswert muss in dem durch den Range-Befehl definierten Bereich liegen und sollte ein Integer oder ein Long-Ausdruck sein.

Rückgabewerte

Ein Roboterpunkt, der durch die angegebenen Pulse bestimmt ist.

Verwandte Befehle

Go, JA, Jump, Move, Pulse Anweisung, XY

Beispiel einer Pulse Funktion

```
Jump Pulse(1000, 2000, 0, 0)
```

Pulse Anweisung

> S

Bewegt den Roboterarm durch PTP-Steuerung zu dem Punkt, der durch die Pulswerte von vier Achsen spezifiziert ist.

Syntax

Pulse [*j1pulses, j2pulses, j3pulses, j4pulses*]

Parameter

j1pulses, j2pulses, j3pulses, j4pulses	Der Pulswert für jede der vier Achsen. Der Pulswert muss in dem durch den Range-Befehl definierten Bereich liegen und sollte ein Integer oder ein Long-Ausdruck sein.

Rückgabewerte

Wenn die Parameter weggelassen werden, werden die vier Achsen-Pulsewerte der aktuellen Armposition angezeigt.

Beschreibung

Pulse verwendet den Achsen-Pulsewert der Pulse-Nullposition, um die Roboterarmposition darzustellen und nicht das orthogonale Koordinatensystem. Der Pulse-Befehl bewegt den Roboterarm durch PTP-Steuerung.

Der Range-Befehl stellte die oberen und unteren Grenzwerte ein, die im Pulse-Befehl verwendet werden können.

Hinweis

Sorgen Sie dafür, dass der Pfad frei von Hindernissen ist, bevor Sie die Pulse-Anweisung verwenden:

Im Gegensatz zu Jump bewegt **Pulse** alle Achsen gleichzeitig, einschließlich der Hub-und Fallbewegung der Z-Achse beim Verfahren zur Zielposition. Daher ist es sehr wichtig, dass Sie bei der Verwendung von **Pulse** darauf achten, dass sich der Arm entlang eines hindernisfreien Pfades bewegen kann.

Potentielle Fehler

Der Pulsewert übersteigt den Grenzwert:

Wenn der mit dem Puls-Befehl spezifizierte Pulswert den durch den Range-Befehl eingestellten Grenzwert überschreitet, tritt ein Fehler auf.

Verwandte Befehle

Go, Accel, Range, Speed, Pls, Pulse Funktion

Beispiel einer Pulse Anweisung

Im Folgenden sehen Sie Beispiele vom Online-Fenster:

Dieses Beispiel bewegt den Roboteram zu der Position, die durch jeden Achsen-Puls definiert wird.

```
> pulse 16000, 10000, -100, 10
```

Dieses Beispiel zeigt die Pulsnummern von der ersten bis zur vierten Achse der aktuellen Roboterarmposition.

```
> pulse
16000 10000
-100  -10
>
```

QP Anweisung

> S

Schaltet den Quick-Pause-Modus Ein oder Aus und zeigt den aktuellen Modusstatus an.

Syntax

QP [{ **On** | **Off** }]

Parameter

On | **Off** Quick Pause kann entweder Ein (On) oder Aus (Off) geschaltet sein.

Rückgabewerte

Zeigt die aktuelle **QP** Moduseinstellung an, wenn der Parameter weggelassen wird.

Beschreibung

Wenn während der Ausführung eines Bewegungsbefehls entweder die Pausetaste gedrückt wird oder ein Pause-Signal an die Steuerung gesendet wird, bestimmt der Quick-Pause-Modus, ob der Roboter sofort angehalten wird oder ob er anhält, nachdem der Bewegungsbefehl durchgeführt wurde.

Sofortiges Verzögern und Anhalten wird "Quick Pause" genannt.

Wenn der Parameter On spezifiziert wurde, schaltet **QP** den Quick-Pause-Modus Ein.
Wenn der Parameter Off angegeben wurde, schaltet **QP** den Quick-Pause-Modus Aus.

QP zeigt die aktuelle Einstellung die entscheidet, ob der Roboterarm auf die Pause-Eingabe reagieren soll, indem er sofort anhält oder nachdem der aktuelle Arbeitsgang des Armes abgeschlossen ist. **QP** ist schlicht ein Status-Befehl, der verwendet wird, um anzuzeigen, ob der Quick-Pause-Modus Ein (On) oder Aus (Off) geschaltet ist.

Hinweis

Der Quick-Pause-Modus wird vorgabemäßig auf On gestellt, nachdem die Versorgungsspannung eingeschaltet wurde.

Der Quick-Pause-Modus, der durch den **QP**-Befehl eingestellt wurde, bleibt nach dem Reset-Befehl bestehen. Wenn jedoch der PC-Strom oder der Strom der Antriebseinheit aus- und dann wieder eingeschaltet wird, wird der Quick-Pause-Modus auf die Vorgabeeinstellung On zurückgesetzt.

QP und der Eingang der Sicherheitsabschrankung:

Auch wenn der **QP**-Modus auf Off geschaltet ist, hält der Roboter sofort an, wenn sich die Sicherheitsabschrankung öffnet.

Verwandte Befehle

Pause, Verinit

Beispiel einer QP Anweisung

Dieses Online-Fenster-Beispiel zeigt die Einstellung, die dem Roboterarm sagt, ob er sofort anhalten soll, wenn Pause eingegeben wird (d.h. der QP-Modus ist Ein (On) oder Aus (Off) geschaltet).

```
> qp
QP ON

> qp On  'Setzt QP aus den Quick-Pause-Modus
>
```

Quit Anweisung

S

Bricht die Ausführung eines bestimmten Tasks oder aller Tasks ab.

Syntax

Quit { *taskIdentifier* | **All** }

Parameter

taskIdentifier	Taskname oder Integer Ausdruck, der die Tasknummer repräsentiert. Ein Taskname ist der Funktionsname, der in einer Xqt-Anweisung verwendet wird, oder eine Funktion, die vom Run- oder vom Benutzer-Fenster aus gestartet wird. Wenn ein Integer Ausdruck verwendet wird, liegt der Bereich zwischen 1 und 32.
All	Bestimmt, dass alle Tasks abgebrochen werden sollen.

Beschreibung

Quit stoppt die Tasks, die gerade ausgeführt werden oder die vorübergehend mit Halt unterbrochen wurden.

Verwandte Befehle

Exit Funktion, Halt, Resume, Xqt

Quit Beispiel

Dieses Beispiel zeigt zwei Tasks, die nach 10 Sekunden abgebrochen werden.

```
Function main
  Xqt winc1   'Startet die Funktion winc1
  Xqt winc2   'Startet die Funktion winc2
  Wait 10
  Quit winc1  'Bricht den Task winc1 ab
  Quit winc2  'Bricht den Task winc2 ab
Fend

Function winc1
  Do
     On 1; Wait 0.2
     Off 1; Wait 0.2
  Loop
Fend

Function winc2
  Do
     On 2; Wait 0.5
     Off 2; Wait 0.5
  Loop
Fend
```

RadToDeg Funktion

> F

»Konvertiert Radiant zu Grad.

Syntax

RadToDeg(*radians***)**

Parameter

radians Real-Ausdruck, welcher die Radianten darstellt, die in Grad konvertiert werden sollen.

Rückgabewerte

Ein Double-Wert, der die Anzahl der Grade enthält.

Verwandte Befehle

ATan, ATan2, DegToRad Funktion

Beispiel einer RadToDeg Funktion

```
s = Cos(RadToDeg(x))
```

Randomize Anweisung

S

Initialisiert den Zufallszahlengenerator.

Syntax

Randomize

Verwandte Befehle

Rnd

Randomize Beispiel

```
Function main
    Real r
    Randomize
    Integer randNum

    randNum = Int(Rnd(10)) + 1
    Print "Random number is:", randNum
Fend
```

Range Anweisung

> S

Spezifiziert die Bewegungs-Grenzen für jede der Servo-Achsen und zeigt sie an.

Syntax

Range [*j1Min, j1Max, j2Min, j2Max, j3Min, j3Max, j4Min, j4Max*]

Parameter

j1Min	Der untere Grenzwert für die erste Achse (T1), angegeben in Pulsen.
j1Max	Der obere Grenzwert für die erste Achse (T1), angegeben in Pulsen.
j2Min	Der untere Grenzwert für die zweite Achse (T2), angegeben in Pulsen.
j2Max	Der obere Grenzwert für die zweite Achse (T2), angegeben in Pulsen.
j3Min	Der untere Grenzwert für die dritte Achse (Z), angegeben in Pulsen.
j3Max	Der untere Grenzwert für die dritte Achse (Z), angegeben in Pulsen.
j4Min	Der untere Grenzwert für die vierte Achse (U), angegeben in Pulsen.
j4Max	Der obere Grenzwert für die vierte Achse (U), angegeben in Pulsen.

Rückgabewerte

Zeigt den aktuellen **Range**-Wert an, wenn **Range** ohne Parameter eingegeben wurde.

Beschreibung

Range bestimmt den unteren und oberen Grenzwert für jede Motorenachse, angegeben in Pulsen. Diese Arbeitsbereichsgrenzen werden in Puls-Einheiten definiert. Das erlaubt es dem Anwender, einen maximalen und einen minimalen Achsen-Bewegungsbereich für jede einzelne Achse zu definieren. XY-Koordinaten-Grenzwerte können auch mit Hilfe des Befehls XYLim eingestellt werden.

Die **Range**-Ausgangswerte sind für jeden Roboter unterschiedlich. Die Werte, die durch diesen Befehl definiert werden, bleiben auch erhalten, nachdem der Strom ausgeschaltet wurde. Der Verinit-Befehl initialisiert diese Werte auf die Vorgabewerte.

Wenn die Parameter weggelassen werden, werden die aktuellen **Range**-Werte angezeigt.

Wichtige Hinweise

Die Range-Einstellung wird durch den Verinit-Befehl nicht beeinflusst:

Weder das Ausschalten der Versorgungsspannung noch der Verinit-Befehl ändern die **Range**-Einstellungen.

Potentielle Fehler

Der Versuch, den zulässigen Bereich zu verlassen:

Wenn der Roboterarm versucht, sich über eine der Arbeitsbereichsgrenzen hinaus zu bewegen, tritt ein Fehler auf.

Die Achse verfährt nicht:

Wenn der untere Pulsgrenze der oberen Pulsgrenze entspricht oder größer als diese ist, bewegt sich die Achse nicht.

Verwandte Befehle

Ver, Verinit, XYLim

Range Beispiel

Das einfache Beispiel vom Online-Fenster zeigt die aktuelle Range-Einstellung an:

```
> range
 -1777     33777
-15555     15555
-13333         0
-25000     25000

> range 0, 32000, 0, 32224, -10000, 0, -40000, 40000
>
```

Read Anweisung

S

Liest Zeichen aus einer Datei oder einer Kommunikationsschnittstelle.

Syntax

Read #fileNumber, stringVar$, count

Parameter

fileNumber	Datei oder Kommunikationsschnittstelle, aus der gelesen werden soll.
stringVar$	Name einer Zeichenkettenvariable, welche die Buchstabenzeichenkette empfängt.
count	Maximale Anzahl der zu lesenden Bytes.

Verwandte Befehle

Seek, Write

Beispiel einer Read Anweisung

```
Integer numOfChars
String data$

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    Read #1, data$, numOfChars
EndIf
```

Real Anweisung

S

Deklariert Real-Variablen (8 Byte Real-Zahl).

Syntax

Real *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]**...**

Parameter

varName	Variablenname, den der Anwender als **Real-**Typ deklarieren will.
subscripts	Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt **(***dim1* [, *dim2* [, *dim3*]]**)** *dim1*, *dim2*, *dim3* können ein Integer zwischen 0 und 32767 sein.

Beschreibung

Real wird verwendet, um Variablen als **Real-**Typ Variablen zu deklarieren. Alle Variablen sollten am Anfang einer Funktion deklariert sein.

Verwandte Befehle

Boolean, Byte, Double, Global, Integer, Long, String

Real Beispiel

Das folgende Beispiel zeigt ein einfaches Programm, das unter Verwendung von **Real** einige Variablen deklariert.

```
Function realtest
  Real var1
  Real A(10)         'Eindimensionale Feldvariable aus Reals
  Real B(10, 10)     'Zweidimensionale Feldvariable aus Reals
  Real C(10, 10, 10) 'Dreidimensionale Feldvariable aus Reals
  Real arrayVar(10)
  Integer i
  Print "Please enter a Real Number:"
  Input var1
  Print "The Real variable var1 = ", var1
  For i = 1 To 5
    Print "Please enter a Real Number:"
    Input arrayVar(i)
    Print "Value Entered was ", arrayVar(i)
  Next i
Fend
```

Redim Anweisung

S

Re-dimensioniert eine Feldvariable (Array) während der Laufzeit.

Syntax

Redim [**Preserve**] *arrayName* **(***subscripts***)**

Parameter

Preserve	Optional. Spezifiziert, den vorherige Inhalt der Feldvariable zu erhalten. Wird die Angabe weggelassen, so wird der Inhalt der Feldvariable gelöscht.
arrayName	Name der Feldvariable; folgt den Standardkonventionen für Variablenbenennung. Die Feldvariable muss bereits deklariert worden sein.
subscripts	Neue Dimensionen der Feldvariable. Sie müssen dieselbe Anzahl von Dimensionen wie bei der Deklaration der Variable angeben. Die Syntax sieht aus wie folgt **(***dim1* [, *dim2* [, *dim3*]]**)** *dim1*, *dim2*, *dim3* kann ein Integer Ausdruck zwischen 0 und 32767 sein.

Beschreibung

Verwenden Sie Redim, um die Dimensionen einer Feldvariable während der Laufzeit zu ändern. Verwenden Sie Preserve, um die vorherigen Werte beizubehalten.

Verwandte Befehle

Erase, UBound

Beispiel einer Redim Anweisung

```
Integer i, numParts, a(0)

Print "Enter number of parts "
Input numParts

Redim a(numParts)

For i=0 to UBound(a)
    a(i) = i
Next

' Re-dimensioniert die Feldvariable mit 20 weiteren Elementen
Redim Preserve a(numParts + 20)

' Die Werte des ersten Elementes werden beibehalten
For i = 0 to UBound(a)
    Print a(i)
Next
```

Rename Befehl

>

Benennt eine Datei um.

Syntax

Rename oldFileName, newFileName

Parameter

oldFileName	Zeichenketten-Ausdruck, der den Pfad und den Namen der umzubenennenden Datei enthält. Dateiname und Dateinamenerweiterung dürfen Platzhalter (*, ?) enthalten.
newFileName	Der neue Name für die Datei, die durch *oldFileName* spezifiziert ist.

Beschreibung

Ändert den Namen der spezifizierten Datei *oldFileName* in *newFileName*.

Wenn der Pfad weggelassen wird, sucht **Rename** im aktuellen Verzeichnis nach *oldFileName*.

Für *newFileName* kann kein Pfad angegeben werden. Daher befindet sich die umbenannte Datei im selben Pfad wie die umzubenennende Datei vorher.

Eine Datei kann nicht umbenannt werden, wenn der gewünschte Name bereits an eine Datei im selben Pfad vergeben ist.

Rename erlaubt die Verwendung von Platzhaltern. Platzhalter, die in filename1 verwendet werden, veranlassen **Rename** dazu, alle entsprechenden Dateien umzubenennen. Um beispielsweise die Namen aller Dateien zu ändern, die von A nach B umbenannt werden sollen, ohne dabei die Endungen der Dateinamen zu ändern, verwenden Sie folgenden Befehl:
Rename A.* B.*

Platzhalter, die in *newFileName* verwendet werden, bewirken, dass **Rename** die *oldFileName*-Zeichen in ihrer jeweiligen Position in *newFileName* beibehält. Um z.B. alle Dateien, die TEST.* entsprechen, nach TEXT.* umzubennenen, verwenden Sie folgenden Befehl:
Rename TEST.* ??X?.*

Verwandte Befehle
Copy
Rename Beispiel

Hier sind einige Beispiele vom Online-Fenster:

> Rename A*.* B*.*
> Rename A.PRG B.PRG

RenDir Befehl

Ändert einen Verzeichnisnamen.

Syntax

Rendir oldDirName, newDirName

Parameter

oldDirName	*Pfad und Name des umzubenennenden Verzeichnisses.*
newDirName	*Der neue Name für das Verzeichnis, das mit oldDirName angegeben ist.*

Beschreibung

Für newDirName kann kein Pfad eingeschlossen sein. Daher befindet sich das umbenannte Verzeichnis im selben Pfad wie das umzubenennende Verzeichnis vorher.

Ein Verzeichnis kann nicht umbenannt werden, wenn der gewünschte Verzeichnisname bereits im selben Pfad vorhanden ist.

Platzhalter sind weder in *oldDirName* noch in *newDirName* gestattet.

Verwandte Befehle
Dir
Beispiel eines RenDir Befehls
Im Folgenden sehen Sie ein einfaches Beispiel vom Online-Fenster.

```
> rendir \BAK  USR2
```

Reset Anweisung

>

Setzt den Controller auf einen initialisierten Status zurück.

Syntax

Reset

Beschreibung

Reset setzt die folgenden Objekte zurück:

Not-Halt-Status
Fehlerstatus
Ausgänge (Alle Ausgänge sind Aus geschaltet; der Anwender kann den Optionsschalter so einstellen, dass dieses Feature ausgeschaltet wird)
Speed und SpeedS für den aktuellen Roboter (werden auf den Vorgabewert zurückgesetzt)
Accel und AccelS für den aktuellen Roboter (werden auf den Vorgabewert zurückgesetzt)
LimZ Paramter des aktuellen Roboters (wird auf 0 zurückgesetzt)
Fine des aktuellen Roboters (wird auf den Vorgabewert zurückgesetzt)
Power Low für den aktuellen Roboter (der Low-Power-Modus wird Ein geschaltet)

Für servo-bedingte Fehler, den Not-Halt-Status und alle anderen Bedingungen, die ein Reset benötigen, wird kein anderer Befehl außer Reset akzeptiert. In diesem Fall führen Sie erst Reset aus, führen Sie dann andere Prozesse wie benötigt aus.

Nach einem Not-Halt z.B. sollten Sie zunächst überprüfen, dass Betriebsbedingungen den Sicherheitsvorschriften entsprechen, dann Reset ausführen und zuletzt den Befehl Motor On.

WICHTIG:

Optionsschalter Reset

Wenn der Optionsschalter “Reset schaltet die Ausgänge Aus” Ein geschaltet ist, werden alle Ausgänge Aus geschaltet, wenn der **Reset**-Befehl ausgeführt wird. Es ist wichtig, dass Sie dies bei der Verdrahtung des Systems bedenken, da das Ausschalten von Ausgängen nicht zum Herunterfallen von Werkzeugen oder ähnlichen Situation führen sollte. Unter dem SPEL-Options-Reiter in der Systemkonfiguration erhalten Sie mehr Details.

Verwandte Befehle

Accel, AccelS, LimZ, Motor, Off, On, SFree, SLock, Speed, SpeedS, Verinit

Beispiel einer Reset-Anweisung

Dieses Beispiel zeigt den Reset-Befehl, vom Online-Fenster aus ausgegeben.

```
>reset
>
```

Resume Anweisung

S

Setzt einen Task fort, der durch den Halt-Befehl unterbrochen war.

Syntax

Resume { *taskIdentifier* | **All** }

Parameter

taskIdentifier	Taskname oder Integer Ausdruck, der die Tasknummer repräsentiert. Ein Taskname ist der Funktionsname, der in einer Xqt-Anweisung verwendet wird, oder eine Funktion, die vom Run- oder vom Operator-Fenster aus gestartet wird. Wenn ein Integer Ausdruck verwendet wird, liegt der Bereich zwischen 1 und 32.
All	Bestimmt, dass alle Tasks fortgesetzt werden sollen.

Beschreibung

Die **Resume** Anweisung setzt die Ausführung der Tasks fort, die durch den Halt-Befehl unterbrochen wurden.

Verwandte Befehle

Halt, Quit, Xqt

Beispiel einer Resume Anweisung

Dieses Beispiel zeigt die Verwendung des Resume-Befehls nach dem Halt-Befehl.

```
Function main
   Xqt 2, flicker   'Führt Flicker als Task 2 aus

   Do
      Wait 3         'Erlaubt, Flicker 3 Sekunden lang auszuführen
      Halt flicker   'Hält den Flicker Task an
      Wait 3
      Resume flicker 'Setzt den Flicker Task fort
   Loop
Fend

Function flicker
   Do
      On 1
      Wait 0.2
      Off 1
      Wait 0.2
   Loop
Fend
```

Restart Anweisung

> S

Startet die aktuelle Programmgruppe neu.

Syntax

Restart

Beschreibung

Restart stoppt alle Tasks und führt die zuletzt gelaufene Programmgruppe erneut aus. Diese Anweisung ist dem Chain-Befehl ähnlich, mit der Ausnahme, dass die Programmgruppe nicht spezifiziert ist. Darüberhinaus kann **Restart** außerdem in einer Trap-Emergency- oder Trap-Fehler-Routine verwendet werden. Das erlaubt es Ihnen, nach einem Not-Halt oder einem kritischen Fehler neu zu starten.

Verwandte Befehle

Chain, Quit, Trap, Xqt

Beispiel einer Restart Anweisung

```
Function main

    Trap Emergency Call tEmer


    If (Stat(0) And 1048576) = 1048576 Then
        Print "Clear Emergency stop"
        While (Stat(0) And 1048576) = 1048576
            Reset
            Wait 1
        Wend
        Print "Emergency stop cleared"
    Endif

    Motor On
    While TRUE
        Call PickPlac
    Wend
Fend

Function tEmer
    Restart
Fend
```

Return Anweisung

S

Die **Return** Anweisung wird zusammen mit der GoSub Anweisung verwendet. GoSub überträgt die Programmsteuerung an ein Unterprogramm. Sobald das Unterprogramm vollständig ausgeführt ist, veranlasst **Return,** dass die Programmausführung in der Zeile fortgesetzt wird, die auf den **GoSub** Befehl folgt, der das Unterprogramm initiiert hatte.

Syntax

Return

Beschreibung

Die **Return** Anweisung wird zusammen mit der Anweisung GoSub verwendet. Der Hauptzweck der **Return** Anweisung ist die Rückgabe der Programmsteuerung an den Befehl, der auf den GoSub-Befehl folgt, der das Unterprogramm ursprünglich ausgelöst hat.

Der GoSub Befehl bewirkt, dass die Programmsteuerung an eine durch den Anwender definierte Anweisungs-Zeilennummer oder zu einem gleichermaßen definierten Label verzweigt wird. Das Programm führt dann die Anweisung in dieser Zeile aus und fährt mit der Ausführung in den darauf folgenden Zeilennummern fort, bis es auf einen **Return**-Befehl trifft. Der **Return**-Befehl veranlasst die Programmsteuerung dann, zu *der* Zeile zurück zu wechseln, welche direkt auf diejenige folgt, welche den GoSub-Befehl ursprünglich initiiert hat. (D.h., der GoSub Befehl veranlasst die Ausführung eines Unterprogramms, wonach die Ausführung durch den **Return-**Befehl zu der Anweisung zurückkehrt, die auf den GoSub Befehl folgt.)

Potentielle Fehler

Return-Befehl ohne GoSub-Befehl gefunden

Ein **Return**-Befehl wird verwendet, um aus einem Unterprogramm in das Ursprungsprogramm zurück zu wechseln, von dem der GoSub Befehl ausgegeben wurde. Wenn ein **Return**-Befehl gefunden wird, ohne dass dem ein GoSub Befehl vorangegangen ist, wird Fehler Nr. 3 ausgegeben. Ein allein stehender **Return**-Befehl ist bedeutungslos, da das System nicht weiß, wohin es zurückkehren soll.

Verwandte Befehle

OnErr, GoSub**,** GoTo

Beispiel einer Return Anweisung

Das folgende Beispiel zeigt eine einfache Funktion, die einen GoSub-Befehl verwendet, um zu einem Label namens checkio zu verzweigen und die 16 ersten Anwender-Eingänge zu überprüfen. Danach kehrt das Unterprogramm zum Hauptprogramm zurück.

```
Function main
     Integer var1, var2
     GoSub checkio
     On 1
     On 2
     End

checkio:  'Hier startet die Subroutine
     var1 = In(0)
     var2 = In(1)
     If var1 <> 0 Or var2 <> 0 Then
          Print "Message to Operator here"
     EndIf
finished:
     Return 'Die Subroutine endet hier und kehrt zu Zeile 40 zurück
Fend
```

Right$ Funktion

F

Gibt eine Zeichenkette der ganz rechts stehenden Zeichen aus, wie vom Anwender spezifiziert.

Syntax

Right$*(string, count)*

Parameter

string	Zeichenkettenvariable oder Buchstabenzeichenkette mit bis zu 255 Zeichen, aus welcher die am weitesten rechts stehenden Zeichen kopiert werden.
count	Anzahl der zu kopierenden Zeichen aus *string,* beginnend mit dem Zeichen, das äußerst rechts steht.

Rückgabewerte

Gibt eine Zeichenkette der äußerst rechts stehenden *count Z*eichen aus der durch den Anwender definierten Buchstabenzeichenkette aus.

Beschreibung

Right$** gibt die äußerst rechts plazierten *count*Zeichen einer durch den Anwender definierten Zeichenkette aus. **Right$ kann so viele Zeichen ausgeben, wie in der Buchstabenzeichenkette vorhanden sind.

Verwandte Befehle

Asc, Chr$, Left$, Len, Mid$, Space$, Str$, Val

Right$ Beispiel

Das Beispiel unten zeigt ein Programm, das eine Part-Zeichenkette als Eingang verwendet und Partnummer, Partnamen und Part-Zählerstand aufteilt.

```
Function SplitPartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    PartNum$ = Left$(DataIn$, 10)

    DataIn$ = Right$(datain$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Mid$(DataIn$, 11, 10)

    PartCount = Val(Right$(dataIn$, 5))

Fend
```

Einige andere Beispielergebnisse des Right$-Befehls vom Online-Fenster.

```
> Print Right$("ABCDEFG", 2)
 FG

> Print Right$("ABC", 3)
 ABC
```

RmDir Befehl

>

Entfernt (löscht) ein leeres Unterverzeichnis aus dem Dateispeicher des Steuergerätes.

Syntax

Rmdir dirName

Parameter

dirName Pfad und Name des zu entfernenden Verzeichnisses.

Beschreibung

Entfernt (löscht) das spezifizierte Unterverzeichnis. Vor der Ausführung von Rmdir müssen alle Dateien des Unterverzeichnisses mit Ausnahme von (.) und (..) gelöscht werden.

Das aktuelle Verzeichnis oder ein übergeordnetes Verzeichnis können nicht gelöscht werden.

Rmdir Beispiel

Im Folgenden sehen Sie ein einfaches Beispiel vom Online-Fenster.

```
> Rd \USR2
```

Rnd Funktion

F

Gibt eine Zufallszahl aus.

Syntax
Rnd(*maxValue*)
Parameter
maxValue Real-Ausdruck, der den maximalen Rückgabewert repräsentiert.

Rückgabewerte
Real-Zufallszahl von 0 bis *range*.
Beschreibung
Verwenden Sie Rnd, um Zufallszahlen-Werte zu generieren.

Verwandte Befehle
Int

Beispiel einer Rnd Funktion
Hier ist ein Rnd-Beispiel, das eine Zufallszahl zwischen 1 und 10 generiert.

```
Function main
   Real r
   Integer randNum

   Randomize
   randNum = Int(Rnd(9)) + 1
   Print "Random number is:", randNum
Fend
```

Robot Anweisung

S

Wählt den aktuellen Roboter aus.

Syntax

Robot *number*

Parameter

number Nummer des gewünschten Roboters. Der Wert reicht von 1 bis zur Anzahl der installierten Roboter.

Beschreibung

Robot ermöglicht es dem Anwender, den Vorgabe-Roboter für nachfolgende Bewegungsbefehle auszuwählen.

Bei einem System mit nur einem Roboter braucht die Robot Anweisung nicht verwendet zu werden.

Verwandte Befehle

Accel, AccelS, Arm, ArmSet, Go, Hofs, Home, HOrdr, HTest, Local, Move, Pulse, Robot Function, Speed, SpeedS

Robot Beispiel

```
Function main
    Integer I
    For I = 1 to 100
        Robot 1
        Go P(i)
        Robot 2
        Go P(i)
    Next I
Fend
```

Robot Funktion

F

Gibt die aktuelle Roboternummer aus.

Syntax

Robot

Rückgabewerte

Integer, welcher die aktuelle Roboternummer enthält.

Verwandte Befehle

Robot Anweisung

Beispiel einer Robot Funktion

```
Print "The current robot is: ", Robot
```

RobotModel$ Funktion

F

Gibt den Namen des Robotermodells aus.

Syntax

RobotModel$ [(*robotNumber*)]

Parameter

robotNumber	Optional. Ein Integer Ausdruck, der eine Roboternummer repräsentiert. Wird diese Angabe weggelassen, so wird die Nummer des aktuellen Roboters verwendet.

Rückgabewerte

Eine Zeichenkette, die den Namen des Modells enthält. Das ist der Name, der auf der Roboterrückseite steht.

Verwandte Befehle

RobotType

RobotModel$ Beispiel

```
Print "The robot model is ", RobotModel$
```

RobotType Funktion

F

Gibt den Robotertyp aus.

Syntax

RobotType [(*robotNumber*)]

Parameter

robotNumber	Optional. Ein Integer Ausdruck, der eine Roboternummer repräsentiert. Wird diese Angabe weggelassen, so wird die Nummer des aktuellen Roboters verwendet.

Rückgabewerte

1: ACHSE

2: KARTESISCH

3: SCARA

4: ZYLINDRISCH

Verwandte Befehle

RobotModel$

RobotType Beispiel

```
If RobotType = 2 Then
  Print "Robot type is Cartesian"
EndIf
```

ROpen Anweisung

S

Öffnet eine Datei zum Lesen.

Syntax

ROpen *fileName* **As** #*fileNumber*

Parameter

fileName	Ein Zeichenkettenausdruck, der den Namen der Datei enthält, aus der gelesen werden soll. Laufwerk und Pfad können ebenfalls enthalten sein.
fileNumber	Integer Ausdruck zwischen 30 und 63, der als Kennung für die Datei verwendet wird.

Beschreibung

Öffnet den spezifizierten Dateinamen zum Lesen und identifiziert ihn durch die spezifizierte *fileNumber*. Diese Anweisung wird verwendet, um eine spezifizierte Datei zu öffnen und Daten aus dieser Datei zu lesen. Der Befehl **Close** schließt die Datei und gibt die Dateinummer aus.

Der spezifizierte Dateiname muss der Name einer auf dem PC existierenden Datei sein.

Die *fileNumber* identifiziert die Datei, solange sie geöffnet ist. Die Datei wird von der Input#-Anweisung zum Lesen und zum Schließen (Close#) verwendet. Entsprechend kann die Dateinummer nicht zur Spezifikation einer anderen Datei verwendet werden, bis die aktuelle Datei geschlossen ist.

Verwandte Befehle

AOpen, Close, Input #, Print #, WOpen

Beispiel einer ROpen Anweisung

Das unten gezeigte einfache Beispiel öffnet eine Datei, schreibt einige Daten hinein, öffnet dieselbe Datei später wieder und liest deren Daten in eine Feldvariable.

```
Real DATA(200)
WOpen "TEST.VAL" As #30
For I = 0 To 100
    Print #30, DATA(i)
Next I
Close #30

ROpen "TEST.VAL" As #30
For I = 0 to 100
    Input #30, DATA(I)
Next I
Close #30
```

RShift Funktion

F

Verschiebt numerische Daten um eine vom Anwender definierte Bitanzahl nach rechts.

Syntax

RShift*(number, shiftBits)*

Parameter

number	Der numerische Ausdruck, der verschoben werden soll.
shiftBits	Bitanzahl, um die die numerischen Daten nach rechts verschoben werden.

Rückgabewerte

Gibt ein numerisches Ergebnis aus, das dem *number*-Wert entspricht, nachdem die Bits um die in *shiftBits* definierte Anzahl von Bits nach rechts verschoben wurden.

Beschreibung

RShift verschiebt die spezifizierten numerischen Daten (*number*) um die festgelegte Bitanzahl (*shiftBits*) nach rechts (an eine geringerwertige Stelle). Die verschobenen Bits höheren Wertes werden durch 0 ersetzt.

Die einfachste Erklärung für **RShift** ist, dass es einfach das Ergebnis von *number* / $2^{shiftBits}$ ausgibt. (*Number* wird dividiert durch 2 *shiftBit* Male.)

Hinweis

Typ numerischer Daten:

Die numerischen Daten (*number*) können aus jeglichem gültigen numerischen Datentyp bestehen. **RShift** arbeitet mit den Datentypen: Byte, Integer und Real.

Verwandte Befehle

And, LShift, Not, Or, Xor

RShift Beispiel

Das unten gezeigte Beispiel zeigt ein Programm, welches alle möglichen **RShift**-Werte für einen Integer-Datentypen aufzeigt, beginnend mit dem auf 0 gesetzten Integer.

```
Function rshiftst
  Integer num, snum, i
  num = 32767
  For i = 1 to 16
     Print "i =", i
     snum = RShift(num, 1)
     Print "RShift(32767, ", i, ") = ", snum
  Next i
Fend
```

Einige andere Beispielergebnisse des **RShift**-Befehls vom Online-Fenster.

```
> Print RShift(10,1)
5
> Print RShift(8,3)
1
> Print RShift(16,2)
4
```

RTrim$ Funktion

F

Gibt eine Zeichenkette aus, die mit der definierten Zeichenkette identisch ist, wobei die nachfolgenden Leerzeichen nicht angeführt werden.

Syntax

RTrim$(*string*)

Parameter

string Zeichenkettenausdruck.

Rückgabewerte

Spezifizierte Zeichenkette, bei der die nachfolgenden Leerzeichen entfernt wurden.

Verwandte Befehle

LTrim$

Beisspiel einer RTrim$ Funktion

```
str$ = "  data  "
str$ = RTrim$(str$)  ' str$ = "..data"
```

RunDialog Anweisung

S

Lässt einen EPSON RC+ Dialog aus einem SPEL+ Programm ablaufen.

Syntax

RunDialog *dialogID*

Parameter

dialogID Integer Ausdruck, der eine gültige Dialog-ID beinhaltet. Diese Werte sind vordefinierte Konstanten, wie unten angezeigt.

DLG_JOG	100	Run Jog and Teach dialog
DLG_IOMON	102	Run I/O Monitor
DLG_ROBOTPANEL	104	Run Robot Control Panel
DLG_VGUIDE	110	Run Vision Guide dialog

Beschreibung

Verwenden Sie RunDialog, um EPSON RC+ Dialoge aus einem SPEL+ Task laufen zu lassen. Der Task wird solange unterbrochen, bis der Operator den Dialog schließt.

Wenn Dialoge laufen, die Roboter-Befehle ausführen, sollten Sie sichergehen, dass keine anderen Tasks den Roboter steuern, während der Dialog angezeigt wird. Anderenfalls tritt ein Fehler auf.

Verwandte Befehle

RunDialog Beispiel

```
If Motor = Off Then
    RunDialog DLG_ROBOTPANEL
    If Motor = Off Then
        Print "Motors are off, aborting program"
        Quit All
    EndIf
EndIf
```

SafetyOn Funktion

F

Gibt den aktuellen Sicherheitsstatus aus.

Syntax
SafetyOn

Rückgabewerte
TRUE, wenn der Sicherheitskreis geöffnet ist (On), ansonsten FALSCH.

Beschreibung

SafetyOn gibt TRUE aus, nachdem der Sicherheitskreis geöffnet worden ist.

SafetyOn entspricht ((Stat(0) And &H400000) <> 0).

Verwandte Befehle
EStopOn, PauseOn, Stat

Beispiel einer SafetyOn Funktion

```
If SafetyOn Then
    Print "Safe Guard is open."
EndIf
```

SavePoints Anweisung

> S

Speichert für den aktuellen Roboter Punktdaten aus dem Hauptspeicher in eine Datei.

Syntax

SavePoints *fileName*

Parameter

fileName	Zeichenkettenausdruck, der die Datei enthält, in welche die Punkte gespeichert werden. Die Dateinamenerweiterung muss .PNT lauten.

Beschreibung

SavePoints speichert die Roboterpunkte in die spezifizierte Datei. Die Erweiterung .PNT muss immer spezifiziert sein.

Der **SavePoints-**Befehl fügt die Punktdatei außerdem dem Projekt des aktuellen Roboters hinzu.

Potentielle Fehler

Speicherkapazität überschritten:

Wenn kein Speicherplatz mehr vorhanden ist, wird Fehler Nr. 60 ausgegeben.

Ungültiger Dateiname

Wird ein Dateiname eingegeben, der mehr als 24 Zeichen lang ist, Leerzeichen beinhaltet oder andere ungültige Dateinamen-Eigenschaften aufweist, wird Fehler Nr. 62 ausgegeben.

Verwandte Befehle

LoadPoints

Beispiel einer SavePoints Anweisung

```
Clear
For i = 1 To 10
   P(i) = i, 100, 0, 0
Next i
SavePoints "TEST.PNT"
```

Seek Anweisung

S

Ändert die Position des Dateizeigers für eine spezifizierte Datei.

Syntax

Seek #*fileNumber*, *pointer*

Parameter

fileNumber	Dateinummer zur Durchführung der Seek-Operation.
pointer	Gewünschte zu suchende Position, beginnend bei 1 bis zur gesamten Länge der Datei.

Verwandte Befehle

Read, Write

Beispiel einer Seek Anweisung

```
Seek #30, 20
Read #30, data$, 2
```

Select ... Send

S

Definiert die Verzweigungsformel sowie die entsprechenden Anweisungssequenzen zum Verzweigen.

Syntax

Select *formula*
Case *item; statement*
.
.
Case *item; statement*
[Default; *statement* **]**
Send

Parameter

formula	Das Vergleichs-Objekt funktioniert dahingehend, dass wenn eines der Case-Objekte gleich *formula* ist, die entsprechende Anweisungen ausgeführt wird. *Formula* kann Konstanten, Variablen, Variablenformeln und Logische Operatoren (das schließt And, Or oder Xor ein) beinhalten.
item	Kann Konstanten oder Variablen beinhalten. Dieses Objekt wird mit jeder Case-Zeile verwendet und mit formula verglichen.
statement	Jeglicher gültige SPEL-Anweisungs-Befehl.

Beschreibung

Wenn irgendein Case-Objekt dem **Select** formula-Ergebnis entspricht, wird diese Case-Objekt-Anweisung ausgeführt. Nach der Ausführung übergibt die Programmsteuerung an den Befehl, der auf Send folgt. (Select End)

Wenn kein Case-Objekt dem **Select** formular-Ergebnis entspricht, wird die Standardanweisung (Default) ausgeführt und die Programmsteuerung wechselt zu dem Befehl, der auf Send folgt.

Wenn Default ausgelassen wird, wird nichts ausgeführt und die Programmkontrolle wechselt zu dem Befehl, der direkt auf Send folgt.

Select kann Konstanten, Variablen, Variablenformeln und Logische Operatoren, die And, Or oder Xor verwenden, beinhalten.

Das Case-Objekt kann Konstanten und Variablen beinhalten. Case-Objekt -Anweisungen können außerdem Mehrfachanweisungen oder Mehrfachzeilen-Anweisungen sein.

Bis zu 32 Case-Objekte sind für 1 **Select**-Anweisung erlaubt.

Verwandte Befehle

If/Then/Else

Select Beispiel

Im Folgenden sehen Sie ein einfaches Beispiel für Select...Send:

```
Function Main
  Integer I
  For I = 0 To 10
    Select I
      Case 0
           Off 1;On 2;Jump P1
      Case 3
           On 1;Off 2
           Jump P2;Move P3;On 3
      Case 7
           On 4
      Default
           On 7
    Send
  Next
Fend
```

Sense Anweisung

> S

Spezifiziert und zeigt die Eingangsbedingung an, die, wenn erfüllt, den gerade in der Ausführung befindlichen JUMP durch anhalten des Roboters über der Zielposition abschließt.

Syntax

Sense [*inputCondition*]

Parameter

Inputcondition **Sense** muss mit der Sw oder Sw($)-Eingabe-Anweisung verwendet werden, um den Zustand der Hardware-Eingänge oder Merker zu überprüfen. Der Zustand muss einen TRUE (-1) oder FALSE (0) –Wert ausgeben. Die folgenden Funktionen und Operatoren können für die *inputCondition* verwendet werden:

Funktionen: Sw, Sw($), In, In($)

Operatoren: And, Or, Xor, +, *

Andere: Klammern, um bestimmten Operationen und Variablen Prioriät einzuräumen.

Beschreibung

Sense wird verwendet, um während eines Jump-Befehls die Abwärtsbewegung der Z-Achse zu stoppen. Die **Sense**-Bedingung muss mindestens 1 Input- oder Merker-Input-Funktion enthalten.

Wenn Variablen in eine Sense-Bedingung eingebunden sind, werden ihre Werte während der **Sense**-Ausführung berechnet. Mehrfache **Sense**-Anweisungen sind erlaubt. Die zuletzt definierte **Sense**-Bedingung bleibt solange aktuell, bis sie von einer anderen **Sense**-Anweisung ersetzt wird.

Jump mit **Sense**-Bedingung:
Überprüft, ob die aktuelle **Sense**-Bedingung erfüllt ist. Wenn die Bedingung erfüllt ist, schließt der Jump-Befehl ab, wenn der Roboter über dem Zielpunkt angehalten hat. (Wenn z.B. die **Sense**-Bedingung 1 ist, bleibt der Roboterarm genau über der Zielposition, ohne die Z-Achse herab zu bewegen.) Wenn die **Sense**-Bedingung 0 ist, beendet der Roboterarm die ganze Jump-Befehl-Bewegung bis zur Zielposition.

Wenn Parameter ausgelassen werden, werden die aktuellen Sense-Definitionen angezeigt.

Hinweis

Sense-Einstellung bei Einschalten der Versorgungsspannung
Beim Einschalten des Stroms lautet die Sense**-Bedingung:**
Sense Sw(0) = 1 Eingang 0 ist eingeschaltet

Verwendung von JS und Stat zur Verifizierung von Sense

Verwenden Sie JS oder Stat, um zu verifizieren, ob die Sense-Bedingung erfüllt wurde.

Verwandte Befehle

Js, Jump, Stat(1), Sw( ), Sw($)

Beispiel einer Sense Anweisung

Dies ist ein einfaches Beispiel für die Verwendung der Sense-Anweisung.

```
Function test
   .
   .
TrySense:
    Sense Sw(1) = 0   'Bestimmt, dass der Arm über
                      'dem Ziel hält, wenn
                      'der Eingang 1 ausgeschaltet ist.
    Jump P1 C2 Sense
    'Wenn der Arm am durch Js(0)=1 definierten Punkt stehen bleibt,
    'wird ERRPRC ausgeführt und das Programm springt zu TrySense.
    If Js(0) = 1 Then
         GoSub ERRPRC
         GoTo TrySense
    EndIf
    On 1; Wait 0.2; Off 1
   .
   .
Fend
```

<Beispiele mit anderer Syntax>

```
> Sense Sw(1)=1 And Sw($1)=1

> Sense Sw(0) Or Sw(1) And Sw($1)
```

SetCom-Anweisung

S

Stellt die Parameter für eine Kommunikationsschnittstelle ein.

Syntax

SetCom #*portNumber,* [*baud* [*, dataBits* [*, stopBits* [*, parity* [*, terminator* [*, HWFlow* [*, SWFlow* [*, timeOut*]]]]]]]]

Parameter

portNumber	Gibt an, für welchen RS232-Port Parameter eingestellt werden sollen. Gültige Einträge sind 1-16.
baud	Optional. Gibt die Baudrate an. Gültige Einträge sind: 110 2400 19200 115200 300 4800 38400 600 9600 56000 1200 14400 57600 (Vorgabeeinstellung: 9600)
dataBits	Optional. Bestimmt die Anzahl der Datenbits pro Zeichen. Gültige Werte sind **7** und **8**.
stopBits	Optional. Bestimmt die Anzahl der Stopbits pro Zeichen. Gültige Werte sind **1** und **2**.
parity	Optional. Bestimmt die Parität. Gültige Werte sind **O** (Odd = ungerade), **E** (Even = gerade) und **N** (None = keine).
terminator	Optional. Bestimmt die Zeilenabschluss-Zeichen. Gültige Werte sind **CR**, **LF**, **CRLF**.
HWFlow	Optional. Bestimmt die Hardwaresteuerung. Gültige Werte sind **RTS** und **NONE**.
SWFlow	Optional. Bestimmt die Softwaresteuerung. Gültige Einträge sind **XON** und **NONE**.
timeOut	Optional. Bestimmt die maximale Zeit für die Übermittlung oder den Empfang, angegeben in Sekunden. Wenn der Wert 0 ist, gibt es kein Timeout.

Hinweise

SetCom ersetzt Config

Bei den SRC 3xx Steuergeräten wurde die Config-Anweisung verwendet, um díe seriellen Anschlüsse der Robotersteuerung zu konfigurieren. Config wird nicht mehr verwendet. Wenn ein Projekt aus einer früheren Version geöffnet wird, die Config verwendet hat, werden die Config-Anweisungen in SetCom-Anweisungen konvertiert.

Verwandte Befehle

OpenCom, CloseCom, SetNet

SetCom Beispiel

```
SetCom #1, 9600, 8, 1, N, CRLF, NONE, NONE, 0
SetCom #2, 4800
```

SetNet Anweisung

S

Stellt die Parameter für einen TCP/IP Port ein.

Syntax

SetNet #*portNumber, hostAddress, TCP_IP_PortNum, terminator, SWFlow, timeOut*

Parameter

portNumber	Gibt an, für welchen Port Parameter eingestellt werden sollen. Gültige Werte sind 128-131.
hostAddress	Gibt die Host IP-Adresse an.
TCP_IP_PortNum	Gibt die TCP/IP Portnummer für diesen Knoten an.
terminator	Gibt die Zeilenabschluss-Zeichen an. Gültige Werte sind **CR**, **LF**, **CRLF**.
SWFlow	Bestimmt die Softwaresteuerung. Gültige Werte sind **XON** und **NONE**.
timeOut	Gibt die die maximale Zeit für die Übermittlung oder den Empfang an, angegeben in Sekunden. Wenn der Wert 0 ist, gibt es kein Timeout.

Verwandte Befehle

OpenCom, CloseCom, SetCom

SetNet Beispiel

```
SetNet #128, 192.168.0.1, 1, CRLF, NONE, 0
```

SFree Anweisung

> S

Schaltet die angegebenen Servoachse frei.

Syntax

SFree *jointNumber* [*, jointNumber,...*]

Parameter

jointNumber	Ein Integer Ausdruck, der eine Servoachsennummer darstellt. Diese kann 1, 2, 3 oder 4 sein, den Achsen 1, 2, Z oder U bei SCARA-Robotern entsprechend.

Beschreibung

SFree schaltet die angegebenen Servoachsen frei. Dieser Befehl wird für das direkte Teachen oder die Teil-Einrichtung verwendet, indem eine bestimmte Achse teilweise abgeschaltet wird. Um eine Achse wieder in Betrieb zu nehmen, führen Sie den SLock-Befehl oder MotorOn aus.

Hinweise

SFree setzt einige System-Objekte zurück auf ihren Anfangsstatus:

SFree setzt aus Sicherheitsgründen Parameter zurück, die die Roboterarmgeschwindigkeit (Speed und SpeedS), die Beschleunigung (Accel und AccelS) und den LimZ-Parameter betreffen.

Wichtig

SFree und seine Verwendung mit der Z-Achse

Die Z-Achse hat elektronische Bremsen. Daher erlaubt es die Einstellung von SFree für die Z-Achse nicht sofort, die Z-Achse zu bewegen. Um die Z-Achse von Hand zu bewegen, muss die Bremse gelöst werden. Betätigen Sie dazu den Schalter der Bremse oben auf dem Roboterarm.

Ausführung von Bewegungsbefehlen während die Achsen im SFree Status sind

Der Versuch, einen Bewegungsbefehl während des SFree Status auszuführen, verursacht einen Fehler im Standardstatus der Steuerung. Um jedoch Bewegungen zu erlauben, solange 1 oder mehrere der Achsen im SFree Status sind, betätigen Sie den Optionsschalter „Bewegung unter sfree ermöglichen“ ("Allow Motion with one or more axes free"). (Dieser Schalter kann über die Roboterkonfigurations-Dialogbox in der Tabelle mit den Optionsschaltern eingestellt werden. Sie erreichen die Dialogbox aus dem Setup-Menü im EPSON RC+.)

Verwandte Befehle

LimZ, Motor, SFree Function, SLock

Beispiel einer SFree Anweisung

Dies ist ein einfaches Beispiel für die Verwendung einer SFree-Anweisung.

```
Function GoPick
   Speed pickSpeed
   'Gibt die Ansteuerung der Achse 1 und
   'Achse 2 frei und kontrolliert die
   'Z-Achse für die Teil-Installation.
   SFree 1, 2
   Go pick
   SLock 1, 2 'Stellt die Ansteuerung der Achsen 1 und 2 wieder her.
Fend
```

SFree Funktion

F

Gibt den SFree Status für eine angegebene Achse aus.

Syntax
SFree(*jointNumber*)
Parameter
jointNumber Integer Ausdruck, der die zu prüfende Achsennummer repräsentiert.
Rückgabewerte
TRUE, wenn die Achse freigeschaltet ist, FALSE, wenn sie nicht freigeschaltet ist.
Verwandte Befehle
SFree Anweisung
SetFree Beispiel

```
If SFree(1) Then
    Print "Joint 1 is free"
EndIf
```

Sgn Funktion

F

Bestimmt das Vorzeichen des Operanden.

Syntax
Sgn(*Operand*)
Parameter
Operand Ein numerischer Ausdruck.

Rückgabewerte
1: Wenn der Operand einen positiven Wert hat.
0: Wenn der Operand 0 ist.
-1: Wenn der Operand einen negativen Wert hat.

Beschreibung
Die **Sgn** Funktion bestimmt das Vorzeichen des numerischen Wertes des Operanden.

Verwandte Befehle
Abs, And, Atan, Atan2, Cos, Int, Mod, Or, Not, Sin, Sqr, Str$, Tan, Val, XOR

Beispiel einer Sgn Funktion
Dies ist ein einfaches Online-Fenster-Beispiel für die Verwendung der Sgn Funktion.

```
>print sgn(123)
 1
>print sgn(-123)
 -1
>
```

Shutdown Anweisung

S

Fährt EPSON RC+ herunter. Windows wird optional heruntergefahren oder neu gestartet.

Syntax

ShutDown [*mode*]

Parameter

mode Optional. Ein Integer Ausdruck, welcher die Moduseinstellungen repräsentiert, die unten beschrieben werden.

Moduseinstellung	Beschreibung
Kein Modus	Zeigt einen Dialog an, der es dem Anwender erlaubt, die Shutdown-Option zu wählen.
0	Fährt EPSON RC+ und Windows herunter.
1	Fährt EPSON RC+ herunter und startet Windows neu.
2	Fährt EPSON RC+ herunter.

Beschreibung

ShutDown ermöglicht es Ihnen, die aktuelle EPSON RC+ Session aus einem SPEL-Programm zu verlassen.

Verwandte Befehle

Restart

Beispiel einer Shutdown Anweisung

```
Exit EPSON RC+ and Windows
ShutDown 0
```

Signal Anweisung

S

Sendet ein Signal an die Tasks, die WaitSig ausführen.

Syntax

Signal *signalNumber*

Parameter

signalNumber Zu übertragende *signalNumber.* Der Bereich liegt zwischen 0 und 127.

Beschreibung

Das Signal kann verwendet werden, um Multi-Task-Ausführungen zu synchronisieren.

Verwandte Befehle

WaitSig

Beispiel einer Signal Anweisung

```
Function Main
   Xqt 2, SubTask
   Call InitSys
   Signal 1

Fend

Function SubTask
   WaitSig 1

Fend
```

Sin Funktion

F

Gibt den Sinus eines numerischen Ausdrucks aus.

Syntax
Sin(*radians***)**
Parameter
radians Real Ausdruck in Radianten.

Rückgabewerte
Numerischer Wert, der den Sinus des numerischen Ausdrucks *radians* repräsentiert.

Beschreibung
Sin gibt den Sinus des numerischen Ausdrucks aus. Der numerische Ausdruck (*radians)* muss in Radianteneinheiten angegeben werden. Der von der **Sin-**Funktion ausgegebenen Wert bewegt sich zwischen -1 und 1.

Verwenden Sie die RadToDeg-Funktion, um Radiantenwerte in Gradzahlen umzuwandeln.

Verwandte Befehle
Abs, Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sqr, Str$, Tan, Val

Beispiel einer Sin Funktion

Das folgende Beispiel zeigt ein einfaches Programm, welches den **Sin**-Befehl nutzt.

```
Function sintest
    Real x
    Print "Please enter a value in radians:"
    Input x
    Print "Sin of ", x, " is ", Sin(x)
Fend
```

SLock Anweisung

> S

Schaltet nach SFree wieder die Leistung auf die angegebene Servoachse.

Syntax

SLock *jointNumber* [*, jointNumber,...*]

Parameter

jointNumber	Die Servoachsennummer. Diese kann 1, 2, 3 oder 4 sein, entsprechend der Achsen 1, 2, Z oder U bei SCARA-Robotern, bzw. der Achsen X, Y, Z oder U bei kartesischen Robotern.

Beschreibung

SLock stellt Servo-Power für die angegebene Servoachse wieder her, die für das direkte Teachen oder die Teileinrichtung durch den SFree-Befehl abgeschaltet war.

Wenn die Achsennummer weggelassen wird, werden alle Achsen aktiviert.

Die Aktivierung der dritten Achse (Z) bewirkt, dass die Bremse gelöst wird.

Um alle Achsen zu aktivieren, kann auch Motor On anstelle von **SLock** verwendet werden.

Die Verwendung von **SLock** während des Motor Off-Status verursacht einen Fehler.

Hinweise

SLock setzt einige System-Objekte zurück auf ihren Anfangsstatus:

SFree setzt aus Sicherheitsgründen Parameter zurück, die die Roboterarmvorschubgeschwindigkeit (Speed und SpeedS), die Beschleunigung (Accel und AccelS) und den LimZ Parameter betreffen.

Verwandte Befehle

LimZ, Reset, SFree

SLock Beispiel

Dies ist ein einfaches Beispiel für die Verwendung einer SLock-Anweisung.

```
Function test
   .
   .
   .
    Sfree 1,2   'Gibt die Ansteuerung der Achse 1
                'Achse 2 frei und kontrolliert die
                'Z- und U-Achse für die Teileinrichtung.
    Go P1
    SLock 1,2   'Gibt die Ansteuerung der
                'Achsen 1 und 2 wieder her.
   .
   .
   .
Fend
```

Space$ Funktion

F

Gibt eine Zeichenkette von *count* Leerzeichen aus, wie vom Anwender spezifiziert.

Syntax

Space$(*count*)

Parameter

count Die Anzahl von Leerzeichen, die in die Ausgabe-Zeichenkette eingesetzt werden sollen.

Rückgabewerte

Gibt eine Zeichenkette von *count* Leerzeichen aus.

Beschreibung

Space$ gibt eine vom Anwender spezifizierte Zeichenkette von *count* Leerzeichen aus. **Space$** kann bis zu 255 Zeichen ausgeben (die maximale Anzahl von Zeichen, die in einer Stringvariable erlaubt ist).

Der **Space$** Befehl wird für gewöhnlich verwendet, um Leerzeichen vor, nach oder zwischen anderen Buchstabenzeichenketten einzufügen.

Verwandte Befehle

Asc, Chr$, Left$, Len, Mid$, Right$, Str$, Val

Beispiel einer Space$ Funktion

```
> Print "XYZ" + Space$(1) + "ABC"
XYZ ABC

> Print Space$(3) + "ABC"
   ABC
>
```

Speed Anweisung

> S

Definiert oder zeigt die Armgeschwindigkeit für die Verwendung mit den PTP-Befehlen Go, Jump und Pulse an.

Syntax

Speed [*percent* [,*ZupSpeed*, *Zdnspeed*]]

Parameter

percent	Integer Ausdruck zwischen 1-100, der die Armgeschwindigkeit als Prozentteil der maximalen Geschwindigkeit darstellt.
ZupSpeed	Integer Ausdruck zwischen 1-100, der die Geschwindigkeit der Aufwärtsbewegung der Z-Achse für den Jump-Befehl darstellt.
ZdnSpeed	Integer Ausdruck zwischen 1-100, der die Geschwindigkeit der Abwärtsbewegung der Z-Achse für den Jump-Befehl darstellt.

Rückgabewerte

Zeigt die aktuellen **Speed**-Werte an, wenn sie ohne Parameter verwendet werden.

Beschreibung

Speed spezifiziert die Armgeschwindigkeit zur Verwendung mit allen PTP-Befehlen. Dies bezieht auch Bewegungen ein, die durch die Roboter-Bewegungsbefehle Go, Jump und Pulse ausgelöst wurden. Die Geschwindigkeit ist als Prozentteil der maximalen Geschwindigkeit spezifiziert. Der Bereich der zulässigen Werte liegt zwischen 1-100 (1 stellt 1% das maximalen Geschwindigkeit dar und 100 stellt 100% der maximalen Geschwindigkeit dar). Speed 100 repräsentiert die maximal mögliche Geschwindigkeit.

Die Werte für die Geschwindigkeit der Aufwärts- und Abwärtsbewegung der Z-Achse beziehen sich ausschließlich auf den Jump-Befehl. Wenn die Werte weggelassen werden, wird jeder Wert auf den *percent* Vorgabewert gesetzt.

Der Speed Wert wird auf seine Vorgabewerte zurückgesetzt, wenn einer der folgenden Befehle ausgeführt wird:

Power On
Software Reset
Motor On
SFree, SLock
Verinit

Im Low-Power-Modus ist die maximale Speed-Einstellung 5% (Vorgabewert). Wenn eine höhere Geschwindigkeit direkt (vom Online-Fenster aus) oder in einem Programm angegeben wurde, wird die Geschwindigkeit auf diesen Vorgabewert gesetzt. Im High-Power-Modus ist die Einstellung von **Speed** gleich dem eingegebenen Wert .

Wenn eine Bewegung bei höherer Geschwindigkeit nötig ist, setzen Sie den High-Power-Modus unter Verwendung von Power High und schließen Sie die Sicherheitstür. Wenn die Sicherheitstür offen ist, werden die **Speed** Einstellungen in ihre Vorgabewerte geändert.

Das folgende Beispiel zeigt, dass der Roboter sich mit der vorgegebenen Geschwindigkeit (5) bewegt, weil er im Low-Power-Modus ist, obschon der eingestellte Geschwindigkeitswert von **Speed** 80 beträgt.

```
> speed 80
> speed
Low Power Mode
    80
    80     80
>
```

Verwandte Befehle

Accel, Go, Jump, Power, Pass, Pulse, SpeedS, TSpeed

Beispiel einer Speed Anweisung

Speed kann vom Online-Fenster aus oder in einem Programm verwendet werden. Unten angegeben sind einfache Beispiele beider Methoden.

```
Function speedtst
    Integer slow, fast, i
    slow = 10
    fast = 100
    For I = 1 To 10
        Speed slow
        Go P0
        Go P1
        Speed fast
        Go P0
        Go P1
    Next I
Fend
```

Vom Online-Fenster aus kann der Anwender auch Speed-Werte einstellen.

```
> Speed 100,100,50   'Die Abwärtsbewegung der Z-Achse wird auf 50 gesetzt
> Speed 50
> Speed
    Low Power State: Speed is limited to 5
       50
       50        50
>
```

Speed Funktion

F

Gibt eine der drei Geschwindigkeitseinstellungen aus.

Syntax

Speed(*paramNumber*)

Parameter

paramNumber	Integer Ausdruck, der einen der unten gezeigten Werte bewertet.
	1: Geschwindigkeit der PTP-Bewegung
	2: Geschwindigkeit der JUMP-Aufwärtsbewegung
	3: Geschwindigkeit der JUMP-Abwärtsbewegung

Rückgabewerte

Integer Wert von 1 bis 100.

Verwandte Befehle

Speed Statement

Beispiel einer Speed Funktion

```
        Integer savSpeed

        savSpeed = Speed(1)
        Speed 50
        Go pick
        Speed savSpeed
    Fend
```

SpeedS Anweisung

> S

Spezifiziert oder zeigt die Armgeschwindigkeit an, die mit den CP-Befehlen wie Move und Arc verwendet wird.

Syntax

SpeedS [*mmPerSec*]

Parameter

mmPerSec	Integer Ausdruck, der die Armgeschwindigkeit in der Einheit mm/s darstellt. Der maximale Wert hängt vom Robotertyp ab.

Rückgabewerte

Zeigt den aktuellen **SpeedS**-Wert an, wenn keine Parameter verwendet werden.

Beschreibung

SpeedS spezifiziert die Werkzeugmittelpunkt-Geschwindigkeit zur Verwendung mit allen CP-Bewegungsbefehlen. Dies schließt auch Bewegungen ein, die durch die Befehle Move und Arc initiiert wurden.

SpeedS wird in mm/sec spezifiziert, was die Werkzeugmittelpunkt-Geschwindigkeit für den Roboterarm repräsentiert. Gültige Einträge für **SpeedS** bewegen sich zwischen 1 und 1120. Der Standardwert variiert von Roboter zu Roboter. Die Standard-SpeedS-Werte für Ihr Robotermodell finden Sie in der Bedienungsanleitung des Roboters. Das ist der **SpeedS**-Ausgangswert, der jedesmal automatisch voeingerichtet wird, wenn die Versorgungsspannung eingeschaltet wird.

Der **SpeedS**-Wert wird auf seine Standardwerte zurückgesetzt, wenn einer der folgenden Befehle ausgeführt wird:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Ctrl + C Key

Im Low-Power-Modus ist die maximale SpeedS-Einstellung 50 mm/s (Vorgabewert). Wenn eine höhere Geschwindigkeit direkt (vom Online-Fenster aus) oder in einem Programm angegeben wurde, wird die Geschwindigkeit auf diesen Vorgabewert gesetzt. Im High-Power-Modus ist die Einstellung von **SpeedS** gleich dem eingegebenen Wert .

Wenn eine Bewegung bei höherer Geschwindigkeit nötig ist, stellen Sie den High-Power-Modus unter Verwendung von Power High ein und schließen Sie die Sicherheitstür. Wenn die Sicherheitstür offen ist, werden die **SpeedS**-Einstellungen in ihre Standardwerte geändert.

Das folgende Beispiel zeigt, dass der Roboter sich mit der vorgegebenen Geschwindigkeit (50) bewegt, weil er im Low-Power-Modus ist, auch wenn der eingestellte Geschwindigkeitswert von **SpeedS** 800 beträgt.

```
> SpeedS 800
Low Power State: SpeedS is limited to 50
>
> SpeedS
Low Power State: SpeedS is limited to 50
  800
>
```

Verwandte Befehle

AccelS, Arc, Move, Speed, TSpeed, TSpeedS

SpeedS Beispiel

SpeedS kann vom Online-Fenster aus oder in einem Programm verwendet werden. Unten angegeben sind einfache Beispiele beider Methoden.

```
Function speedtst
    Integer slow, fast, i
    slow = 50
    fast = 500
    For I = 1 To 10
        SpeedS slow
        Go P0
        Move P1
        SpeedS fast
        Go P0
        Move P1
    Next I
Fend
```

Vom Online-Fenster aus kann der Anwender auch **SpeedS**-Werte setzen.

```
> speeds 1000
> speeds 500
> speed 30     'Setzt die Point-to-Point-Geschwindigkeit
> go p0        'Point-to-Point-Bewegung
> speeds 100   'Setzt die geradlinige Geschwindigkeit in mm/s
> move P1      'Geradlinige Bewegung
```

SpeedS Funktion

F

Gibt die aktuelle SpeedS Einstellung aus.

Syntax

SpeedS

Rückgabewerte

Real Zahl, in mm/s

Verwandte Befehle

SpeedS Statement

SpeedS Beispiel

```
Real savSpeeds

    savSpeeds = SpeedS
```

SPELCom_Event Anweisung

> S

Generiert ein Anwender Event für ein VB-Guide SPELCom-Steuerelement, die in einem Host-Programm verwendet wird.

Syntax

SPELCom_Event *eventNumber* [*, msgArg1, msgArg2, msgArg3,...*]

Parameter

eventNumber	Ein Integer Ausdruck, dessen Wert zwischen 1000 und 32767 liegt.
msgArg1, msgArg2, msgArg3...	Optional. Jedes Nachrichtenargument kann entweder eine Zahl, eine Buchstabensymbol-Zeichenkette oder ein Variablenname sein.

Beschreibung

Dieser Befehl erleichtert es, Echtzeitinformationen an eine andere Anwendung zu senden, die die SPELCom ActiveX-Steuerung verwenden. Diese wird von der VB-Guide-Option bereitgestellt. Sie können z.B. den Teile-Zählerstand, die Losmenge usw. aktualisieren, indem Sie ein Ereignis an Ihr Host-Programm schicken.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die VB-Guide-Option installiert ist.

Verwandte Befehle

VB Guide Manual

Beispiel eines SPELCom_Event

In diesem Beispiel sendet ein SPEL-Task Zyklusdaten an das Host-Programm.

```
Function RunParts
    Integer cycNum

    cycNum = 0
    ' Hauptschleife
    While True
        ...
        ...
        cycNum = cycNum + 1
        Spelcom_Event 3000, cycNum, lot$, cycTime
    Wend
Fend
```

Verwandte Befehle

VB Guide Manual

SPELCom_Return Anweisung

> S

Gibt einen Wert von einer SPEL-Funktion an die Call-Methode eines VB Guide SPELCom-Steuerelements aus.

Syntax

SPELCom_Return *returnValue*

Parameter

returnValue	Ein Integer Ausdruck, der von der SPELCom Call-Methode zurückgesendet wird.

Beschreibung

SPEL Com_Return ist ein komfortabler Weg, um einen Wert an die SPELCom Host-Anwendung auszugeben. Wenn **SPELCom_Return** ausgeführt wird, ohne von SPELCom aufgerufen worden zu sein, wird der Befehl ignoriert und es tritt kein Fehler auf.

Hinweis

Dieser Befehl ist nur dann anwendbar, wenn die VB-Guide-Option installiert ist.

SPELCom_Return Beispiel

In diesem Beispiel ruft ein VB-Programm die SPEL-Funktion *GetPart* auf, welche eine Fehlernummer ausgibt.

```
'VB-Code
Dim sts As Integer
sts = SPELCom1.Call("GetPart")


'SPEL-Code
Function GetPart
    Integer errNum
    OnErr GPErr
    errNum = 0
    Jump P1
    On vacuum
    Wait Sw(vacOn) = 1, 2
    If Tw = 1 Then
        errNum = VAC_TIMEOUT
    Endif
GPExit:
    SPELCom_Return errNum
    Exit Function
GPErr:
    errNum = Err
    EClr
    GoTo GPExit
Fend
```

Sqr Funktion

F

Berechnet den nicht-negativen Quadratwurzelwert des Operanden.

Syntax
Sqr(*Operand*)
Parameter
Operand Ein Ausdruck mit einer Real Zahl.

Rückgabewerte
Quadratwurzelwert.
Beschreibung
Die Sqr Funktion gibt den nicht-negativen Quadratwurzelwert des Operanden aus.

Potentielle Fehler

Negativer Operand:
Wenn der Operand ein negativer numerischer Wert ist oder einen solchen besitzt, tritt ein Fehler auf.

Verwandte Befehle
Abs, And, Atan, Atan2, Cos, Int, Mod, Not, Or, Sgn, Sin, Str$, Tan, Val, Xor

Beispiel einer Sqr Funktion

Dies ist ein einfaches Online-Fenster-Beispiel für die Verwendung der Sqr Funktion.

```
>print sqr(2)
 1.414214
>
```

Das folgende Beispiel zeigt ein einfaches Programm, das den **Sqr**-Befehl nutzt.

```
Function sqrtest
    Real x
    Print "Please enter a numeric value:"
    Input x
    Print "The Square Root of ", x, " is ", Sqr(x)
Fend
```

Stat Funktion

F

Gibt Informationen über den Ausführungsstatus der Steuerung aus.

Syntax

Stat*(address)*

Parameter

address Definiert, welche Status-Bits überprüft werden sollen.

Rückgabewerte

Gibt einen 4 Byte-Wert aus, der den Status der Steuerung darstellt. Siehe Tabelle unten.

Beschreibung

Der **Stat** Befehl gibt Informationen aus, wie in der Tabelle unten gezeigt.

Adresse	Bit		Angegebener Status der Steuerung, wenn das Bit eingeschaltet ist.
0	0 bis 15	&H0 bis &H8000	Task 1 wird ausgeführt (Xqt) oder ist im Halt-Status bis Task 16 wird ausgeführt (Xqt) oder ist im Halt-Status
	16	&H10000	Task(s) wird/werden ausgeführt.
	17	&H20000	Pausezustand
	18	&H40000	Fehlerzustand
	19	&H80000	Attend-Modus
	20	&H100000	Not-Halt-Zustand
	21	&H200000	Low-Power-Modus (Energie ist niedrig)
	22	&H400000	Eingang der Sicherheitsabschrankung ist geschlossen.
	23	&H800000	Freigabeschalter ist offen.
	24-31		Nicht definiert
1	0	&H0	Protokolleintragung des Stopps über der Zielposition bei erfüllter Bedingung in der Jump...Sense Anweisung. (Dieser Protokolleintrag wird gelöscht, wenn eine andere Jump-Anweisung ausgeführt wird).
	1	&H1	Protokolleintragung des Stopps an einer Zwischenposition bei erfüllter Bedingung in der Go/Jump/Move...Till Anweisung. (Dieser Protokolleintrag wird gelöscht, wenn eine andere Go/Jump/Move...Till-Anweisung ausgeführt wird).
	2	&H2	Protokolleintrag der Ausführung des Mcal-Befehls / der Mcal-Anweisung.
	3	&H4	Protokolleintrag des Stopps an einer Zwischenposition bei erfüllter Bedingung in der Trap Anweisung.
	4	&H8	Motor On-Modus
	5	&H20	Die aktuelle Position ist die Home-Position.
	6	&H40	Low Power-Status
	7	&H80	Nicht definiert
	8	&H100	Der Motor der 4. Achse ist eingeschaltet.
	9	&H200	Der Motor der 3. Achse ist eingeschaltet.
	10	&H400	Der Motor der 2. Achse ist eingeschaltet.
	11	&H800	Der Motor der 1. Achse ist eingeschaltet.
	12-31		Nicht definiert
2	0 bis 15	&H0 bis &H8000	Task 17 wird ausgeführt (Xqt) oder ist im Halt-Status bis Task 32 wird ausgeführt (Xqt) oder ist im Halt-Status

Verwandte Befehle

EStopOn Function, TillOn Function, PauseOn Function, SafetyOn Function

Stat Beispiel

```
Function StatDemo

    rbt1_sts = RShift((Stat(0) And &H070000), 16)
    Select TRUE
        Case (rbt1_sts And &H01) = 1
            Print "Tasks are running"
        Case (rbt1_sts And &H02) = 2
            Print "Pause Output is ON"
        Case (rbt1_sts And &H04) = 4
            Print "Error Output is ON"
    Send
Fend
```

Str$ Funktion

F

Konvertiert einen numerischen Wert in eine Zeichenkette. Gibt eine Zeichenkette aus.

Syntax

Str$*(number)*

Parameter

number Integer oder Real Ausdruck.

Rückgabewerte

Gibt eine Zeichenkettendarstellung des numerischen Wertes aus.

Beschreibung

Str$ konvertiert eine Zahl in eine Zeichenkette. Jede Zahl mit bis zu 18 Zeichen (positiv oder negativ) ist gültig.

Verwandte Befehle

Abs, Asc, Chr$, Int, Left$, Len, Mid$, Mod, Right$, Sgn, Space$, Val

Beispiel einer Str$ Funktion

Das folgende Beispiel zeigt ein Programm, welches mehrere verschiedene Zahlen in Zeichenketten umwandelt und diese dann am Bildschirm ausgibt.

```
Function strtest
    Integer intvar
    Real realvar
    '
    intvar = -32767
    Print "intvar = ", Str$(intvar)
    '
    realvar = 567.9987
    Print "realvar = ", Str$(realvar)
    '
Fend
```

Einige andere Beispielergebnisse des **Str$**-Befehls vom Online-Fenster.

```
> Print Str$(99999999999999)
 1.000000E+014

> Print Str$(25.999)
 25.999
```

String Anweisung

S

Deklariert String-Variablen. (Zeichenketten-Variablen)

Syntax

String *varName$* [**(***subscripts***)**] [, *varName$* [**(***subscripts***)**]**...**

Parameter

varName$	Variablenname, den der Anwender als **String** deklarieren will.
subscripts	Optional. Dimensionen einer Feldvariable; es können bis zu drei multiple Dimensionen deklariert werden. Die Syntax sieht aus wie folgt **(***dim1* [, *dim2* [, *dim3*]]**)** *dim1*, *dim2*, *dim3* können eine ganze Zahl zwischen 0 und 32767 sein.

Beschreibung

String wird verwendet, um Variablen als **String** zu deklarieren. Variablen des Typs **String** können bis zu 255 Zeichen beinhalten. Alle Variablen sollten ganz oben in einer Funktion deklariert sein.

String Operatoren

Die folgenden Operatoren können verwendet werden, um **String**-Variablen zu manipulieren:

+ Fügt Buchstaben-**String**s zusammen**.** Kann in Zuweisungskommandos für **String**-Variablen oder im Print-Befehl verwendet werden.

```
Beispiel: name$ = fname$ + " " + lname$
```

= Vergleicht Buchstaben-**String**s. TRUE wird nur ausgegeben, wenn die beiden Zeichenketten völlig gleich sind.

```
Beispiel: If temp1$ = "A" Then GoSub test
```

< > Vergleicht Buchstaben-STRINGs. TRUE wird ausgegeben, wenn ein oder mehrere Zeichen in den beiden Zeichenketten unterschiedlich sind.

```
Beispiel: If temp1$ <> "A" Then GoSub test
```

Hinweise

Variablennamen müssen das Dollarzeichen "$" enthalten:
Bei Variablen des **String**-Typs muss an letzter Stelle des Variablennamens das Dollarzeichen "$" stehen.

Verwandte Befehle
Boolean, Byte, Double, Global, Integer, Long, Real

String Beispiel

```
String password$
String A(10)         'Eindimensionale Feldvariable aus Zeichenketten
String B(10, 10)     'Zweidimensionale Feldvariable aus Zeichenketten
String C(10, 10, 10) 'Dreidimensionale Feldvariable aus Zeichenketten

Print "Enter password:"
Input password$
If UCase$(password$) = "EPSON" Then
    Call RunMaintenance
Else
    Print "Password invalid!"
EndIf
```

Sw Funktion

F

Gibt den Status des gewählten Eingangsports aus oder zeigt ihn an. (z.B. einzelner Eingang)

Syntax

Sw(*bitNumber***)**

Parameter

bitNumber Zahl zwischen 0 und 511, die einen einzelnen Standard- oder Erweiterungs-Hardware-Eingang darstellt.

Rückgabewerte

Gibt eine 1 aus, wenn der spezifizierte Eingang eingeschaltet ist und eine 0, wenn der spezifizierte Eingang ausgeschaltet ist.

Beschreibung

Sw bietet Ihnen die Möglichkeit, eine Statusüberprüfung der Hardwareeingänge durchzuführen. **Sw** wird im allgemeinen verwendet, um den Status eines Einganges zu überprüfen, welcher an einen Feeder, ein Förderband, einen Greifermagneten oder an den Host eines anderen Gerätes angeschlossen sein sollte, welches über einzelne E/A arbeitet. Natürlich hat der mit dem **Sw**-Befehl überprüfte Eingang 2 Status (1 oder 0). Diese zeigen an, ob das Gerät ein- oder ausgeschaltet ist.

Einschränkung

Der Unterschied zwischen Sw und Sw $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischen den Befehlen **Sw** und Sw $ versteht. Der **Sw $** -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf den Hardware E/As. Der **Sw**-Befehl arbeitet mit den Hardware-Eingangskanälen, die sich auf der Rückseite des Steuergerätes befinden. Diese Hardwarekanäle sind einzelne Eingänge, die mit Geräten außerhalb des Steuergerätes kommunizieren.

Verwandte Befehle

In, In $, InBCD, Off, Off $, On, On OpBCD, Oport, Out, Out $, Sw $, Wait

Beispiel einer Sw Funktion

Das Beispiel unten überprüft lediglich den einzelnen Eingang Nr. 5 und verzweigt dem entsprechend. Zur besseren Übersicht wird On anstelle von 1 verwendet.

```
Function main
    Integer i, feed5rdy
    feed5rdy = Sw(5)
    'Überprüft, ob der Feeder bereit ist
    If feed5rdy = On Then
        Call mkpart1
    Else
        Print "Feeder #5 is not ready. Please reset and"
        Print "then restart program"
    EndIf
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> print Sw(5)
1
>
```

Sw($)

F

Gibt den Status des gewählten Merker-Bits aus oder zeigt ihn an.

Syntax

Sw(*$bitNumber***)**

Parameter

bitNumber Zahl zwischen 0 und 511, die einen der 512 Merker darstellt. HINWEIS: Das Dollarzeichen "$" muss vor die Bitnummer gestellt werden, um anzuzeigen, dass dies ein Merker ist.

Rückgabewerte

Gibt eine 1 aus, wenn das spezifizierte Bit eingeschaltet ist und eine 0, wenn das spezifizierte Bit ausgeschaltet ist.

Beschreibung

Sw $ bietet eine interne Merker-Fähigkeit. Sw $ wird im allgemeinen als Statusvariable verwendet, die 2 Status hat (1 oder 0), welche an/aus, wahr/falsch, erledigt/nicht erledigt usw. bedeuten können. Gültige Einträge für Sw $ liegen im Bereich von Bit 0 bis Bit 511. Normalerweise werden die Befehle On $ und Off $ mit dem Befehl Sw $ verwendet. On $ schaltet das angegebene Bit ein und Off $ schaltet das angegebene Bit aus.

Einschränkung

Der Unterschied zwischen Sw und Sw $

Es ist sehr wichtig, dass der Anwender den Unterschied zwischen den Befehlen **Sw** und Sw $ versteht. Der Off $ -Befehl arbeitet mit dem internen Merker und hat keinerlei Auswirkungen auf die Hardware E/As des Systems. Der **Sw**-Befehl arbeitet mit den Hardware-Eingangskanälen, die sich auf der Rückseite des Steuergerätes befinden. Diese HARDWAREkanäle sind einzelne Eingänge, die mit Geräten außerhalb des Steuergerätes kommunizieren.

Verwandte Befehle

In, In$, InBCD, Off, Off$, On, On$, OpBCD, Oport, Out, Out$, Sw, Wait

Sw$ Beispiel

Das folgende Beispiel zeigt 2 Tasks. Jeder der beiden Tasks hat die Möglichkeit, Bewegungsbefehle zu initiieren. Jedoch wird ein Sicherungsmechanismus zwischen den beiden Tasks verwendet, um sicherzustellen, dass ein Task erst dann die Kontrolle über die Bewegungsbefehle des Roboters erhält, wenn der andere Task deren Verwendung abgeschlossen hat. Dies ermöglicht es 2 Tasks, Bewegungsbefehle korrekt und in geordneter, vorhersehbarer Art und Weise auszuführen. Sw $ wird in Kombination mit dem Wait-Befehl verwendet, um zu warten, bis Merker Nr. 1 den richtigen Wert erreicht hat, von dem an es sicher ist, eine neue Bewegung auszuführen.

```
Function main
    Integer I
    Off $1
    Xqt 2, task2
    For I = 1 to 100
        Wait Sw($1) = 0
        Go P(i)
        On $1
    Next I
Fend

Function task2
    Integer I
    For I = 101 to 200
        Wait Sw($1) = 1
        Go P(i)
        Off $1
    Next I
Fend
```

Andere einfache Beispiele vom Online-Fenster sehen aus wie folgt:

```
> on $1
> print sw($1)
1
> off $1
> print sw($1)
0
```

Tab$ Funktion

F

Weiterbewegen der aktuellen Print-Position um eine spezifizierte Anzahl von Tabulatoren.

Syntax

Tab$*(number)*

Parameter

number Integer Ausdruck, der die Anzahl der Tabulatoren darstellt.

Rückgabewerte

Beschreibung

Eine Zeichenkette, welche die spezifizierte Anzahl von Tabulatoren beinhaltet.

Verwandte Befehle

Left$, Mid$, Right$, Space$

Beispiel einer Tab$ Funktion

```
Print "X", Tab$(1), "Y"
Print
For i = 1 To 10
    Print x(i), Tab$(1), y(i)
Next i
```

Tan Funktion

F

Gibt den Tangens eines numerischen Ausdrucks aus.

Syntax

Tan(*radians***)**

Parameter

radians Real Ausdruck, angegeben in Radianten.

Rückgabewerte

Real Zahl, die den Tangens des Parameters *radians* beinhaltet.

Beschreibung

Tan gibt den Tangens des numerischen Ausdrucks aus. Der numerischer Ausdruck (*radians*) kann ein beliebiger numerischer Wert sein, solange dieser in Radianten-Einheiten angegeben wird.

Verwenden Sie die RadToDeg-Funktion, um Radiantenwerte in Gradzahlen umzuwandeln.

Verwandte Befehle

Abs, Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sin, Sqr, Str$, Val

Beispiel einer Tan Funktion

```
Function tantest
    Real num
    Print "Enter number in radians to calculate tangent for:"
    Input num
    Print "The tangent of ", num, "is ", Tan(num)
Fend
```

Das folgende Beispiel zeigt einige typische Resultate der Verwendung des Tan Befehls vom Online-Fenster aus.

```
> print tan(0)
0.00
> print tan(45)
1.6197751905439
>
```

TaskDone Funktion

F

Syntax

TaskDone (*taskIdentifier*)

Parameter

taskIdentifier	Taskname oder Integer Ausdruck, der die Tasknummer repräsentiert. Ein Taskname ist der Funktionsname, der in einer Xqt-Anweisung verwendet wird, oder eine Funktion, die vom Run- oder vom Operator-Fenster aus gestartet wird. Wenn ein Integer Ausdruck verwendet, liegt der Bereich zwischen 1 und 32.

Rückgabewerte

TRUE, wenn der Task abgeschlossen wurde, FALSE, wenn er nicht abgeschlossen wurde.

Beschreibung

Verwenden Sie TaskDone um zu ermitteln, ob ein Task abgeschlossen wurde.

Verwandte Befehle

TaskState, TaskWait

Beispiel einer TaskDone Funktion

```
Xqt 2, conveyor
Do
    .
    .
Loop Until TaskDone(conveyor)
```

TaskState Funktion

F

Syntax

TaskState(*taskIdentifier*)

Parameter

taskIdentifier	Taskname oder Integer Ausdruck, der die Tasknummer repräsentiert. Ein Taskname ist der Funktionsname, der in einer Xqt-Anweisung verwendet wird, oder eine Funktion, die vom Run- oder vom Operator-Fenster aus gestartet wird. Wenn ein Integer Ausdruck verwendet wird, liegt der Bereich zwischen 1 und 32.

Rückgabewerte

0: Der Task läuft nicht.
1: Der Task läuft.
2: Der Task wartet auf ein Ereignis.
3: Der Task wurde angehalten.
4: Der Task ist mit QuickPause angehalten worden.
5: Der Task befindet sich im Fehlerzustand.
6: Der Task führt die Wait-Anweisung durch.

Beschreibung

Verwenden Sie TaskState, um den Status für den angebenen Task zu erhalten. Sie können die Tasknummer oder den Tasknamen spezifizieren.

Verwandte Befehle

TaskDone, TaskWait

Beispiel einer TaskState Funktion

```
If TaskState(conveyor) = 0 Then
    Xqt 2, conveyor
EndIf
```

TaskWait Anweisung

S

Wartet darauf, dass ein Task beendet wird.

Syntax

TaskWait (*taskIdentifier*)

Parameter

taskIdentifier	Taskname oder Integer Ausdruck, der die Tasknummer repräsentiert. Ein Taskname ist der Funktionsname, der in einer Xqt-Anweisung verwendet wird, oder eine Funktion, die vom Run- oder vom Operator-Fenster aus gestartet wird. Wenn ein Integer Ausdruck verwendet wird, liegt der Bereich zwischen 1 und 32.

Verwandte Befehle

TaskDone, TaskState

Beispiel einer TaskWait Anweisung

```
Xqt 2, conveyor
TaskWait conveyor
```

TGoAnweisung

> S

Führt eine relative PTP- Bewegung im aktuellen Werkzeug-Koordinatensystem aus.

Syntax

TGo *destination* [**CP**] [*searchExpr*] [*!...!*]

Parameter

destination	Der Zielort der Bewegung, unter Verwendung eines Punktausdrucks.
CP	Optional. Spezifiziert die Continuous Path (CP / kontinuierlicher Weg)-Bewegung.
searchExpr	Optional. Ein Till- oder Find-Ausdruck. **Till** | **Find** **Till Sw**(*expr*) = {**Ein** | **Aus**} **Find Sw**(*expr*) = {**Ein** | **Aus**}
!...!	Optional. Parallelbearbeitungsanweisungen können hinzugefügt werden, um I/O und andere Befehle während der Bewegung auszuführen.

Beschreibung

Führt eine relative PTP- Bewegung im ausgewählten Werkzeug-Koordinatensystem aus.

Die Till-Bedingung wird verwendet, um TGo durch Verzögerung und Stoppen des Roboters an einer Zwischenposition des Verfahrweges abzuschließen, wenn die aktuelle Till-Bedingung erfüllt wird.

Der Find-Befehl wird verwendet, um einen Punkt in FindPos zu speichern, wenn die Find-Bedingung während der Bewegung erfüllt wird.

Verwandte Befehle

Accel, Find, !....! Parallel Processing, Point Assignment, Speed, Till, TMove, Tool

TGo Beispiel

Im Folgenden sehen Sie ein einfaches Beispiel vom Online-Fenster.

```
> TGo 100, 0, 0, 0      'Bewegt das aktuell gewählte Werkzeug um 100mm
in die X-Richtung
                        '(im Werkzeugkoordinatensystem)
```

Till Anweisung

> S

Spezifiziert und zeigt die Eingangsbedingung an, die, wenn erfüllt, den durchgeführten Bewegungsbefehl (Jump, Go, Move usw.) abschließt, indem der Roboter an einer Zwischenposition verzögert und angehalten wird.

Syntax

Till [*inputCondition*]

Parameter

inputCondition **Till** muss mit der Sw oder Sw($)-Eingabe-Anweisung verwendet werden, um die Bedingung der Hardware-Eingänge oder Merker zu überprüfen Der Zustand muss einen WAHR (1) oder FALSCH (0) –Wert ausgeben. Die folgenden Funktionen und Operatoren können für die *inputCondition* verwendet werden:

Funktionen: Sw, Sw($), In, In($), Force

Operatoren: And, Or, Xor, +, *

Andere: Klammern, um bestimmten Operationen und Variablen Priorität einzuräumen.

Beschreibung

Die **Till**-Bedingung muss mindestens 1 Input- oder Merker-Input-Funktion enthalten. Fügen Sie in die **Till**-Bedingung nur die Operatoren ein, die im vorangegangenen Abschnitt "Parameter" beschrieben werden. (Die Verwendung eines anderen Operators erzeugt zwar keinen Fehler, führt aber zu unvorhersehbaren Bewegungen.)

Die **Till** Anweisung kann in einer separaten Zeile oder als Suchausdruck in einer Bewegungsbefehl-Anweisung verwendet werden.

Wenn Variablen mit einbezogen sind, werden ihre Werte während der **Till**-Ausführung berechnet. Multiple **Till**-Anweisungen sind erlaubt. Die zuletzt aufgetretene **Till**-Bedingung bleibt solange aktuell, bis sie ersetzt wird.

Wenn der Parameter weggelassen wird, werden die aktuellen Till-Definitionen angezeigt.

Hinweis

Till-Einstellung bei Einschalten der Versorgungsspannung:

Wenn die Stromzufuhr eingeschaltet wird, wird die **Till**-Bedingung auf **Till** Sw(0)=1 initialisiert.

Verwenden Sie Stat, um Till zu verifizieren:

Nach dem Ausführen eines Bewegungsbefehles, der die **Till**-Bedingung verwendet, kann es Fälle geben, in denen Sie verifizieren möchten, ob die **Till**-Bedingung erfüllt wurde oder nicht. Das können Sie tun, indem Sie die Stat Funktion verwenden.

Verwandte Befehle

Find, Go, Jump, Move, Stat, Sw, Sw($), TillOn

Till Beispiel

Unten angegeben sind einige Zeilen eines Programms, das die Till-Anweisung verwendet.

```
Till Sw(1) = 0   'Definiert die Till-Bedingung (Eingang 1 aus)
Go P1 Till     'Stoppt, wenn die Bedingung der vorigen Zeile erfüllt ist
Till Sw(1) = 1 And Sw($1) = 1   'Definiert die neue Till-Bedingung
Move P2 Till   'Stoppt, wenn die Bedingung der vorigen Zeile erfüllt ist
Move P5 Till Sw(10) = 1      'Stoppt, wenn die Bedingung in dieser Zeile
                             'erfüllt ist
```

TillOn Funktion

F

Gibt den aktuellen Till-Status aus.

Syntax
TillOn

Rückgabewerte
TRUE, wenn die Till-Bedingung im vorangegangenen Bewegungsbefehl, der Till verwendet hat, aufgetreten ist.

Beschreibung
TillOn gibt WAHR aus, wenn die Till Bedingung aufgetreten ist.

TillOn entspricht ((Stat(1) And 2) <> 0).

Verwandte Befehle
EStopOn, SafetyOn, Stat

Beispiel einer TillOn Funktion

```
Go P0 Till Sw(1) = 1
If TillOn Then
    Print "Till condition occurred during move to P0"
EndIf
```

Time Befehl

>

Spezifiziert die aktuelle Zeit und zeigt sie an.

Syntax

Time [*hours, minutes, seconds*]

Parameter

hours	Die Stunde des Tages, auf welche die Uhr der Steuerung eingestellt werden soll. Integer Ausdruck zwischen 1 und 24.
minutes	Die Minute des Tages, auf welche die Uhr der Steuerung eingestellt werden soll. Integer Ausdruck zwischen 0 und 59.
seconds	Die Sekunde des Tages, auf welche die Uhr der Steuerung eingestellt werden soll. Integer Ausdruck zwischen 0 und 59.

Rückgabewerte

Wenn die Parameter weggelassen werden, wird die aktuelle Zeit im 24-Stunden-Format angezeigt.

Beschreibung

Spezifiziert die aktuelle Zeit.

Die Zeit wird im 24-Stunden-Format angegeben.

Verwandte Befehle

Date, GetTime, GetDate

Time Beispiel

Hier sind einige Beispiele vom Online-Fenster:

```
Time
The current time is 10:15:32

> Time 1,5,0
> Time
The current time is 1:05:15
```

Time Funktion

F

Gibt die Betriebsstunden der Robotersteuerung aus.

Syntax

Time(*unitSelect*)

Parameter

unitSelect Ein Integer zwischen 0 und 2. Dieser Integer spezifiziert, welche Zeiteinheit das Steuergerät ausgibt:

0: Stunden

1: Minuten

2: Sekunden

Beschreibung

Gibt die Betriebsstunden der Robotersteuerung als Integer aus.

Verwandte Befehle

Hour

Beispiel einer Time Funktion

Hier sind einige Beispiele vom Online-Fenster:

```
Function main
    Integer h, m, s

    h = Time(0)   'Speichert die Zeit in Stunden
    m = Time(1)   'Speichert die Zeit in Minuten
    s = Time(2)   'Speichert die Zeit in Sekunden
    Print "This controller has been used:"
    Print h, "hours, ",
    Print m, "minutes, ",
    Print s, "seconds"
Fend
```

Time$ Funktion

F

Gibt die Systemzeit aus.

Syntax

Time$

Rückgabewerte

Eine Zeichenkette, welche die aktuelle Zeit im 24-Stunden-Format *hh:mm:ss* enthält.

Beschreibung

Time$ wird verwendet, um die Systemzeit in einer Programmanweisung zu erhalten. Um die Systemzeit einzustellen, müssen Sie den Time-Befehl vom Online-Fenster aus ausführen.

Verwandte Befehle

Date, Date$, GetDate, GetTime, Time

Time$ Beispiel

```
Function LogErr

   Integer errNum

   errNum = Err(0)
   AOpen "errlog.dat" As #30
   Print #30, "Error" , errNum, ErrMsg$(errNum), " ", Date$, " ", Time$
   Close #30
   EClr
Fend
```

TLSet Anweisung

> S

Definiert ein Werkzeugkoordinatensystem oder zeigt es an.

Syntax

TLSet [*toolNum*, *Xposition, Yposition, Zposition, RotAngle*]

Parameter

toolNum	Integer von 1 bis 3, die darstellt, welches von 3 Werkzeugen (Tools) definiert werden soll. (Werkzeug 0 ist das Vorgabewerkzeug und kann nicht modifiziert werden.)
Xposition	Ursprung der X-Position des Werkzeugkoordinatensystems.
Yposition	Ursprung der Y-Position des Werkzeugkoordinatensystems.
Zposition	Ursprung der Z-Position des Werkzeugkoordinatensystems.
RotAngle	Rotationswinkel des Werkzeugkoordinatensystems

Rückgabewerte

Wenn die Parameter weggelassen werden, wird die aktuelle **TLSet** Definition angezeigt.

Beschreibung

Definiert die Werkzeugkoordinatensysteme Tool 1, Tool 2 und Tool 3 durch die Spezifizierung des Ursprungs und des Rotationswinkels des Werkzeugkoordinatensystems in Relation zum Koordinatensystem des Tool 0 (Hand-Koordinaten-System).

TLSet 1,50,100,-20,30

TLSet 2,P10 +X(20)

Im zweiten Fall wird Bezug auf die Koordinaten-Werte von P10 genommen und zu dem X-Wert wird 20 addiert. Arm-Attribut und Nummern des lokalen Koordinatensystems werden ignoriert.

Hinweise

Die TLSet-Werte werden beibehalten

Die **TLSet**-Werte werden dauerhaft gespeichert und nicht geändert, bis sie entweder vom Anwender geändert werden, oder ein Verinit-Befehl ausgegeben wird.

Verwandte Befehle

Tool, Arm, ArmSet, TLSet Function

TLSet Beispiel

Das folgende Beispiel zeigt einen guten Test, der vom Online-Fenster ausgeführt werden kann. Er hilft dabei, den Unterschied zwischen Bewegungen mit einem Tool und ohne ein Tool zu verstehen.

```
> TLSet 1, 100, 0, 0, 0  'Definiert das Werkzeugkoordinatensystem für
                         'Tool 1 (plus 100 mm in X-Richtung
                         'vom Arm-Koordinatensystem)
> Tool 1  'Wählt Tool 1 wie durch TLSet definiert
> TGo P1  'Positioniert die Spitze des Werkzeugs 1 auf P1
> Tool 0  'Sagt dem Roboter, dass er für zukünftige Bewegungen kein Tool
           verwenden soll
> Go P1   'Positioniert den Mittelpunkt der U-Achse auf P1
```

TLSet Funktion

F

Gibt einen Punkt aus, der die Werkzeugdefinition für das spezifizierte Tool enthält.

Syntax

TLSet(*toolNumber*)

Parameter

toolNumber	Integer Ausdruck, der die Nummer der auszugebenden Werkzeugdefinition darstellt.

Rückgabewerte

Ein Punkt, der die Werkzeugdefinition beinhaltet.

Verwandte Befehle

TLSet Statement

Beispiel einer TLSet Funktion

```
P1 = TLSet(1)
```

TMOut Anweisung

> S

Spezifiziert die Anzahl von Sekunden, die auf die Erfüllung der mit dem Wait-Befehl spezifizierten Bedingung gewartet wird, bevor ein Timeout-Fehler ausgegeben wird.

Syntax

TMOut *seconds*

Parameter

seconds Integer Ausdruck der darstellt, wie viele Sekunden abzuwarten sind, bis ein Timeout eintritt. Der gültige Bereich ist 0 bis 32767 Sekunden, in 1-Sekunden-Intervallen.

Beschreibung

TMOut setzt die Zeit fest, die gewartet werden soll (wenn der Wait-Befehl verwendet wird), bis ein Timeout-Fehler (Fehler Nr. 94) auftritt. Wenn ein Timeout von 0 Sekunden spezifiziert ist, ist das Timeout ausgeschaltet. In diesem Fall wartet der Wait-Befehl auf unbestimmte Zeit, bis die festgelegte Bedingung erfüllt ist.

Der vorgegebene Erstwert für **TMOut** ist 0.

Verwandte Befehle

In, IN, OnErr, Sw, Sw$, Wait

TMOut Beispiel

```
TMOut 5
Wait Sw($0)
If Tw Then
    Print "Time out occurred"
EndIf
```

TMove Anweisung

> S

Führt im gewählten Werkzeug-Koordinatensystem eine linearinterpolierte Relativbewegung aus.

Syntax

TMove *destination* [**CP**] [*searchExpr*] [*!...!*]

Parameter

destination	Der Zielort der Bewegung, unter Verwendung eines Punktausdrucks.
CP	Optional. Spezifiziert die Continuous Path (CP / kontinuierlicher Weg)-Bewegung.
searchExpr	Optional. Ein Till- oder Find-Ausdruck. **Till** | **Find** **Till Sw**(*expr*) = {**Ein** | **Aus**} **Find Sw**(*expr*) = {**Ein** | **Aus**}
!...!	Optional. Parallelbearbeitungsanweisungen können hinzugefügt werden, um E/A und andere Befehle während der Bewegung auszuführen.

Beschreibung

Führt im gewählten Werkzeug-Koordinatensystem eine linearinterpolierte Relativbewegung aus.

Die Till-Bedingung wird verwendet, um TMove durch Verzögerung und Stoppen des Roboters an einer Zwischenposition des Verfahrweges abzuschließen, wenn die aktuelle Till-Bedingung erfüllt ist.

Der Find-Befehl wird verwendet, um einen Punkt in FindPos zu speichern, wenn die Find-Bedingung während der Bewegung erfüllt wird.

Verwandte Befehle

AccelS, Find, !....! Parallel Processing, Point Assignment, SpeedS, TGo, Till, Tool

TMove Beispiel

Im Folgenden sehen Sie ein einfaches Beispiel vom Online-Fenster.

```
> TMove 100, 0, 0, 0     'Bewegt das aktuell gewählte Werkzeug um 100mm
                         in die 'X-Richtung (im
                         Werkzeugkoordinatensystem)
```

Tmr Funktion

F

Timer-Funktion, welche die Zeit, die abgelaufen ist seit der Timer gestartet wurde, in Sekunden angibt.

Syntax

Tmr(*timerNumber*)

Parameter

timerNumber	Integer Ausdruck der darstellt, von welchem der 64 Timer die Zeit überprüft werden soll.

Rückgabewerte

Laufzeit des angegebenen Timers, dargestellt als Real Zahl in Sekunden. Der Timer deckt den Bereich von 0 bis ca. 1.7E+31 ab. Die Timer-Auflösung beträgt 0.000001 Sekunden.

Beschreibung

Gibt die Zeit, die vergangen ist seit der spezifizierte Timer gestartet wurde, in Sekunden aus. 64 Timer sind verfügbar, nummeriert mit 0 – 63.

Timer werden mit TmReset auf Null gestellt.

SPEL+ Timer verwenden die hochauflösenden Leistungszähler der CPU des PCs. Wenn sehr kleine Zeitintervalle ausgewertet werden, beachten Sie, dass es eine geringe Systemverwaltungszeit beim Abfragen der Timerwerte aus Ihrem SPEL+ Programm gibt. Sie können die Systemverwaltungszeit berechnen, indem Sie Tmr direkt nach TMReset aufrufen. Da die Systemverwaltungszeit sehr klein ist (sie beträgt ca. 10 Mikrosekunden), ist es normalerweise nicht erforderlich, dies in Betracht zu ziehen.

```
Real overhead

TmReset 0
overHead = Tmr(0)
```

Verwandte Befehle

TmReset

Beispiel einer Tmr Funktion

```
TmReset 0          'Setzt den Timer 0 zurück
For I=1 To 10      'Führt Operationen 10 mal durch
    GoSub Cycle
Next
Print Tmr(0) / 10  'Berechnet die Taktzeit und zeigt sie an
```

TmReset Anweisung

> S

Der Timer wird unter Verwendung der Tmr Funktion auf Null zurückgesetzt.

Syntax

TmReset *timerNum*

Parameter

timerNum	Integer Ausdruck von 0 – 63, der darstellt, welcher der 64 Timer zurückgesetzt werden soll.

Beschreibung

Setzt den Timer auf Null zurück und startet den Timer unter Verwendung von *timerNum*. 64 Timer sind verfügbar, nummeriert von 0 bis 63.

Verwenden Sie die Tmr Funktion, um die Laufzeit für einen spezifizierten Timer auszugeben.

Verwandte Befehle

Tmr

TmReset Beispiel

```
TmReset 0           'Setzt den Timer 0 zurück
For I=1 To 10       'Führt Operationen 10 mal durch
    GoSub CYL
Next
Print Tmr(0)/10     'Berechnet die Taktzeit und zeigt sie an
```

Tool Anweisung

> S

Wählt eine Tool (Werkzeug)-Definition oder zeigt diese an.

Syntax

Tool [*toolNumber*]

Parameter

toolNumber Integer Ausdruck von 0 bis 3, der darstellt, welche von 4 Werkzeugdefinitionen mit den anstehenden Bewegungsbefehlen verwendet werden soll.

Rückgabewerte

Zeigt die aktuellen **Tool**-Werte ohne Verwendung von Parametern an.

Beschreibung

Tool wählt das Werkzeug aus, das durch die Werkzeugnummer (toolNum) spezifiziert ist. Wenn die Werkzeugnummer 0 ist, ist kein Werkzeug ausgewählt und alle Bewegungen werden in Bezug auf den Mittelpunkt des U-Achsen-Flansches durchgeführt. Wenn jedoch der Werkzeugeintrag 1, 2 oder 3 ausgewählt ist, wird die Bewegung in Bezug auf das Ende des Werkzeuges, wie in der Werkzeugdefinition definiert, durchgeführt.

Hinweis

Abschalten der Versorgungsspannung und die Auswirkungen auf die Werkzeugwahl:

Das Ausschalten der Versorgungsspannung ändert die Werkzeugkoordinatensystem-Wahl nicht. Jedoch setzt die Ausführung des Verinit Befehls die Werkzeugkoordinatensytem-Wahl auf das Standardwerkzeug (Tool 0) zurück.

Verwandte Befehle

TGo, TLSet, TMove

Beispiel einer Tool Anweisung

Das folgende Beispiel zeigt einen guten Test, der vom Online-Fenster ausgeführt werden kann. Er hilft dabei, den Unterschied zwischen Bewegungen mit einem Tool und ohne ein Tool zu verstehen.

```
>tlset 1, 100, 0, 0, 0  'Definiert das Werkzeugkoordinatensystem für
                         'Werkzeug 1 (plus 100 mm in X-Richtung
                         'vom Arm-Koordinatensystem)
>tool 1    'Wählt Werkzeug 1 wie durch TLSet definiert
>tgo P1    'Positioniert die Spitze des Werkzeugs 1 auf P1
>tool 0    'Sagt dem Roboter, dass er für zukünftige Bewegungen kein
            Werkzeug verwenden soll
>go P1     'Positioniert den Mittelpunkt der U-Achse auf P1
```

Tool Funktion

F

Gbit die aktuelle Werkzeugnummer aus.

Syntax

Tool

Rückgabewerte

Integer, welcher die aktuelle Werkzeugnummer beinhaltet.

Verwandte Befehle

Tool Anweisung

Beispiel einer Tool Funktion

```
Integer savTool

savTool = Tool
Tool 2
Go P1
Tool savTool
```

Trap Anweisung

S

Definiert Traps und was passieren soll, wenn sie eintreten.

Syntax

(1) **Trap** *trapNumber*, *inputCondition* **GoTo** {*linenum* | *label*}
Trap *trapNumber*, *inputCondition* **GoSub** {*linenum* | *label*}
Trap *trapNumber*, *inputCondition* **Call** *funcname*
Trap *trapNumber*

(2) **Trap Emergency Call** *funcName*
Trap Error Call *funcName*
Trap Pause Call *funcName*
Trap SGOpen Call *funcName*
Trap SGClose Call *funcName*
Trap Abort Call *funcName*

Parameter

trapNumber	Integer von 1 bis 4, die darstellt, welcher von 4 Trapnummern verwendet werden soll. (SPEL unterstützt bis zu 4 aktive Traps zur selben Zeit.)
inputCondition	Diese Bedingung muss einen TRUE (1) oder FALSE (0) –Wert ausgeben. Die folgenden Funktionen und Operatoren können für die *inputCondition* verwendet werden: **Funktionen:** Sw, Sw($), In, In($) **Operatoren:** And, Or, Xor, +, * **Andere:** Klammern, um bestimmten Operationen und Variablen Prioriät einzuräumen.
lineNumber	Die Zeilennummer, an welche die Programmausführung übertragen wird, wenn die Befehle GoTo oder GoSub mit Trap verwendet werden und die Trap-Bedingung erfüllt ist.
label	Die Marke, an welche die Programmausführung übertragen wird, wenn die Befehle GoTo oder GoSub mit Trap verwendet werden und die Trap-Bedingung erfüllt ist.
funcname	Die Funktion, die aufgerufen wird, wenn Call mit den Trapbefehlen verwendet wird und die Trap-Bedingung erfüllt ist.

Beschreibung

Grundsätzlich gibt es zwei Arten von Traps. Die eine verwendet Syntax 1 und ist für Hardware- oder Merker-Eingänge vorgesehen, die andere verwendet die Syntax 2 und ist für verschiedene Systembedingungen vorgesehen.

Wenn ein Trap auftritt, der Call verwendet, müssen Sie den Trap durch Ausführung einer Trap-Anweisung am Ende der Trap-Funktion erneut definieren. Wenn Sie den Trap wieder erneut definieren bevor die Funktion endet und die Trap-Bedingung auftritt, bevor die erste Trap-Funktion abgeschlossen ist, tritt ein Fehler auf.

Sie sollten den Code in den Trap-Behandlungsroutinen so klein wie möglich halten.

Syntax 1

Führt einen Trapprozess aus, welcher durch GoTo, GoSub oder Call spezifiziert ist, wenn die spezifizierte Eingangsbedingung erfüllt ist.
Sobald der Trap-Prozess ausgeführt wurde, ist die **Trap**-Einstellung gelöscht. Wenn derselbe Interrupt-Prozess noch einmal benötigt wird, muss der **Trap**-Befehl ihn erneut definierten.

Wenn eine Eingangsbedingung erfüllt wird, während eine andere Funktion durch den Call-Befehl ausgeführt wird, wird der Trap-Prozess durch GoTo, GoSub in der **Trap**-Einstellung nicht ausgeführt.

Um eine **Trap**-Einstellung zu löschen, führen Sie einfach den **Trap**-Befehl nur mit dem *trapNumber* Parameter durch. Beispielsweise löscht „Trap 3“ den Trap Nr. 3.

Wenn GoTo spezifiziert ist
Der Befehl, der ausgeführt wird, wird wie unten beschrieben verarbeitet. Dann verzweigt die Steuerung an die spezifizierte Zeilennummer oder das spezifizierte Label.
- Jede Armbewegung hält unverzüglich an.
- Der Warte-Status durch die Warte- oder Eingabebefehle wird nicht fortgesetzt.
- Alle anderen Befehle beenden die Ausführung bevor die Steuerung verzweigt.

Wenn GoSub spezifiziert ist
Nach der Ausführung desselben Prozesses wie bei GoTo, wird an die spezifizierte Zeilennummer oder das spezifizierte Label verzweigt. Dann wird das darauf folgende Unterprogramm ausgeführt. Sobald die Return-Anweisung am Ende des Unterprogramms ausgeführt ist, kehrt die Programmausführung zurück zu der Zeile, die auf den GoSub-Befehl folgt.

Die Befehle GoSub und Call sind im Unterprogramm eines Trap-Prozesses nicht erlaubt.

Wenn Call spezifiziert ist
Die Programmsteuerung führt die angegebene Funktion aus. In diesem Fall arbeitet der Task, der den Trapbefehl ausführt, weiter.

Syntax 2
Während Not-Halt, Schutztür geöffnet oder geschlossen, Fehler oder Pause wird die Trap-Prozess-Funktion des Call-Befehls als privilegierter Task ausgeführt.

Verwenden Sie Xqt nicht in der Trap-Prozess-Funktion.

Um eine Trap-Einstellung zu löschen, führen Sie den Trap-Befehl nur mit dem Schlüsselwort-Parameter aus. Zum Beispiel löscht “Trap Emergency” den Not-Halt-Trap.

Wenn Emergency spezifiziert ist
- Wenn der Not-Halt aktiviert ist, führt die Programmsteuerung die Trap-Prozess-Funktion aus, nachdem der Shutdown-Prozess für den Task beendet ist. Das bedeutet, dass die Trap-Routine der letzte auszuführende Code ist, nachdem alle Tasks gestoppt haben. Jedoch ist der Restart-Befehl verfügbar, den Sie in der Trap-Emergency-Behandlungsroutine verwenden können, um die aktuelle Gruppe neu zu starten. Sie können auch den Chain-Befehl verwenden, um einen spezifizierte Gruppe zu starten.
- Wenn der E/A-Status so konfiguriert ist, dass er beim Erkennen von Not-Halt auf Null zurückgesetzt wird, können die Anweisungen On, Off und Out für E/A in der Trap-Prozess-Funktion nicht ausgeführt werden.
- Wenn der E/A-Status so konfiguriert ist, dass er beibehalten werden soll, wenn Not-Halt erkannt wird, sind alle E/A-Anweisungen verfügbar. Dieses Feature wird über einen der Option-Schalter eingestellt oder deaktiviert. Um das Feature zu konfigurieren, gehen Sie bitte auf den SPEL-Options-Tab im System-Konfigurationsdialog, den Sie über das Setup Menü erreichen können.

Wenn Error spezifiziert ist
- Wenn ein Fehler (einschließlich einem Fehler, der innerhalb des Systems aufgetreten ist) ausgegeben wird, führt das System die Trap-Prozess-Funktion aus, nachdem der Shutdown-Prozess für den Task beendet ist. Das bedeutet, dass die Trap-Routine der letzte auszuführende Code ist, nachdem alle Tasks gestoppt haben. Jedoch ist der Restart-Befehl verfügbar, den Sie in Trap-Emergency-Behandlungsroutine verwenden können, um die aktuelle Gruppe neu zu starten. Sie können auch den Chain-Befehl verwenden, um eine spezifizierte Gruppe zu starten.
- Es werden nur Fehler erkannt, die dazu führen, dass ein Task beendet wird. Zum Beispiel wird der Fehler-Trap-Funktion aufgerufen, wenn ein Task keine Fehlerbehandlungsroutine hat und ein Fehler auftritt.

Wenn Pause spezifiziert ist
- Wenn eine Pause eintritt, während ein Programm ausgeführt wird, wird die Trap-Prozess-Funktion ausgeführt, nachdem der Pausen-Status verarbeitet wurde. Der Pause-Zustand kann eintreten, wenn eine Pause-Anweisung ausgeführt wird, die Schutztür offen ist oder bei Remote Pause.

Wenn SGOpen spezifiziert ist
- Wenn die Schutztür geöffnet wird, während ein Programm läuft, wird eine Trap-Prozess-Funktion ausgeführt, nachdem der Pausen-Status verarbeitet wurde.

Wenn SGClose spezifiziert ist
- Wenn durch eine geöffnete Schutztür eine Pause erzeugt wird, während ein Programm ausgeführt wird, und diese Tür dann wieder geschlossen wird, wird die Trap-Prozess-Funktion ausgeführt.

Wenn Abort spezifiziert ist
- Wenn ein AbortAll vom Run-Window aus eintritt, oder Stop vom Operator-Window, dann wird diese Trap-Funktion aufgerufen. Diese Funktion sollte kurz sein und keine Endlosschleife beinhalten. Wenn während dieser Unterbrechung ein zweiter AbortAll auftritt, werden alle Tasks abgebrochen und der Trap wird nicht aufgerufen.

Verwandte Befehle

Call, Era, Erl, Err, Ert, ErrMsg$, GoSub, GoTo, OnErr

Trap Beispiel

<Beispiel 1> Fehlerprozess durch den Anwender definiert.

Sw(0) Eingang wird als anwenderdefinierter Fehlereingang betrachtet.

```
Function Main
    Trap 1 Sw(0)=On GoTo EHandle  'Definiert Trap
    .
    .
    .
EHandle:
    On 31   'Signalampellicht
    OpenCom #1
    Print #1, "Error is issued"
    CloseCom #1
Fend
```

<Beispiel 2> Verwendungsweise wie beim Multi-Tasking

```
Function Main
    Trap 2 Sw($0)=1 Or Sw($1)=1 Call Feeder
    .
    .
    .
Fend
.
Function Feeder
  Select TRUE
    Case Sw($0) = On
      Off $0
      On 2
    Case Sw($1) = On
      Off $1
      On 3
  Send

  ' Rüstet den Trap für das nächste Mal auf
  Trap 2, Sw($0) = On Or Sw($1) = On Call Feeder
Fend
```

<Beispiel 3> Not-Halt-Behandlungsroutine

```
Wenn der Eingang Sw(31) eingeschaltet ist, wird der Wait-Status
unterbrochen und die Ausführung zum Trap-Prozess-Unterprogramm
verzweigt.
Function Main
    Trap Emergency Call EstopHandler
    .
    Print "Starting main"
    Do
       Print "Main is running"
       Wait 1
    Loop
    Exit Function

Fend

Function EstopHandler
  Print "EStop occurred"
  Print "Clear the EStop to continue"
  Do
    Reset
  Loop Until Not EStopOn
  Print "Restarting program"
  Restart
Fend
```

<Beispiel 4> Behandlungsroutine für kritische Fehler

```
Function Main
    Trap Error Call CriticalHandler
    Print "Starting main"
    Do
       Wait 1
       ' Kein Fehlerbehandlungsprogramm in dieser Funktion
       ' Daher löst die nächste Zeile den
       ' Error-Trap aus
       Print 1 / 0
    Loop
    Exit Function
Fend

Function CriticalHandler
  Print "Critical error occurred"
  Print "Restarting program"
  Restart
Fend
```

Tw Funktion

F

Gibt den Status der Befehle Wait, WaitNet und WaitSig aus.

Syntax

Tw

Rückgabewerte

Wenn die Wait-Bedingung erfüllt ist, wird 0 ausgegeben.
Wenn das Zeitintervall abgelaufen ist, wird 1 ausgegeben.

Beschreibung

Die Timer Wait-Funktion **Tw** gibt den Status der vorangehenden Wait-Bedingung mit Zeitintervall als 0 (Wait-Bedingung erfüllt) oder 1 (Zeitintervall abgelaufen) aus.

Verwandte Befehle

Wait

Beispiel einer Tw Funktion

```
Wait Sw(0) = On, 5 'Wartet bis zu 5 Sekunden darauf, dass Eingang 0
eingeschaltet wird
If Tw = 1 Then
    Goto TIME_UP   'Verzweige nach TIME_UP nach 5 Sekunden
EndIf
```

Type Befehl

>

Zeigt den Inhalt der angegebenen Datei an.

Syntax

Type *fileName*

Parameter

fileName Pfad und Name der anzuzeigenden Datei.

Beschreibung

Type zeigt die Inhalte der angegebenen Datei an. Da nur ASCII-Dateien angezeigt werden können, stellen Sie bitte sicher, dass nur ASCII-Dateien spezifiziert sind. Der Zweck von Type ist es, den Inhalt von Dateien anzuzeigen und nicht, Dateien zu bearbeiten.

ASCII-Dateien

Quellprogramm (.PRG)
Punktdaten-Datei (.PNT)
Dateien, die durch ROpen oder WOpen erzeugt wurden

Verwandte Befehle

List

Type Beispiel

Schreiben Sie den Inhalt einer Textdatei.

```
> type test.dat
MyData Line 1
MyData Line 2
MyData Line 3
>
```

UBound Funktion

F

den größten verfügbaren Index für die angegebene Dimension einer Matrix aus.

Syntax Gibt
UBound *(arrayName [, dimension])*

Parameter

arrayName	Name der Feldvariable; folgt den Standardkonventionen für Variablenbenennung.
dimension	Optional. Integer Ausdruck, der angibt, von welcher Dimension die Obergrenze ausgegeben wird. Verwenden Sie 1 für die erste Dimension, 2 für die zweite und 3 für die dritte. Wenn *dimension* weggelassen wird, dann wird von Dimension 1 ausgegangen.

Verwandte Befehle
Erase, Redim

UBound Function Example

```
Integer i, a(10)

For i=0 to UBound(a)
    a(i) = i
Next
```

Ucase$ Funktion

F

Gibt eine in Großbuchstaben umgewandelte Zeichenkette aus.

Syntax

UCase$ (*string*)

Parameter

string Zeichenkettenausdruck.

Rückgabewerte

Die konvertierte Großbuchstaben-Zeichenkette.

Verwandte Befehle

LCase$

Ucase$ Beispiel

```
str$ = "Data"
str$ = UCase$(str$)  ' str$ = "DATA"
```

Val Funktion

F

Konvertiert eine aus Zahlen bestehende Buchstabenzeichenkette in ihren numerischen Wert und gibt diesen Wert aus.

Syntax

Val*(string)*

Parameter

string Zeichenkettenausdruck, der nur aus numerischen Zeichen besteht.

Rückgabewerte

Gibt ein Integer oder Fließkomma-Ergebnis aus, abhängig von der Zeichenketten-Eingabe. Wenn die Zeichenketten-Eingabe ein Dezimalpunkt-Zeichen enthält, wird die Zahl in eine Fließkommazahl umgewandelt. Anderenfalls ist der Rückgabewert ein Integer.

Beschreibung

Val konvertiert eine Buchstabenzeichenkette in einen numerischen Wert. Das Ergebnis kann ein Integer oder eine Fließkommazahl sein. Wenn die Zeichenkette der Val-Instruktion, welche übergeben wird, ein Dezimalzeichen enthält, wird eine Fließkommazahl zurückgegeben. Anderenfalls wird ein Integer ausgegeben.

Verwandte Befehle

Abs, Asc, Chr$, Int, Left$, Len, Mid$, Mod, Right$, Sgn, Space$, Str$

Val Beispiel

Das folgende Beispiel zeigt ein Programm, welches mehrere verschiedene Zeichenketten in Zahlen umwandelt und diese dann auf dem Bildschirm ausgibt.

```
Function ValDemo
    String realstr$, intstr$
    Real realsqr, realvar
    Integer intsqr, intvar

    realstr$ = "2.5"
    realvar = Val(realstr$)
    realsqr = realvar * realvar
    Print "The value of ", realstr$, " squared is: ", realsqr

    intstr$ = "25"
    intvar = Val(intstr$)
    intsqr = intvar * intvar
    Print "The value of ", intstr$, " squared is: ", intsqr
Fend
```

Das folgende Beispiel wird vom Online-Fenster ausgeführt.

```
> Print Val("25.999")
25.999
>
```

Ver Befehl

>

Zeigt die Systemkonfigurationsparameter an.

Syntax

Ver

Rückgabewerte

Zeigt die Systemkonfigurationsparameter an.

Beschreibung

Zeigt die aktuell definierten Werte der Systemsteuerungsdaten an. Nach Lieferung des Roboters oder bei Änderung der Daten sollten diese Daten gesichert werden. Diese Sicherung können Sie mit dem Befehl Mkver aus dem Wartungsdialog durchführen. Außerdem sollten Sie diese Daten ausdrucken und an einem sicheren Ort aufbewahren, da Informationen wie Kalibrierungsdaten und Roboter-Spezifikations-Informationen mit dem Ver-Befehl angezeigt werden. Die Rückgabewerte des Ver-Befehls können aus dem Druck-Dialog in EPSON RC+ gedruckt werden, indem der Ausgabe-Dialog des Ver-Befehls markiert wird.

Die folgenden Daten werden angezeigt. (Die folgenden Daten sind nur zur Bezugnahme, da die Daten von Steuergerät zu Steuergerät variieren.)

```
' Version:
'   EPSON RC+ 3.0.0

' Options:
'   SPEL Runtime Drivers
'   Vision Guide
'   VB Guide

' HOUR: 5.887

' Drive Unit 1:
'   MIB I/O Address: 300
'   MIB Mem Address: D8000
'   Motor 1: Enabled, Power = 800, Gain = F7F6, Offset = FE00
'   Motor 2: Enabled, Power = 400, Gain = FDFD, Offset = 200
'   Motor 3: Enabled, Power = 100, Gain = FDFC, Offset = FF04
'   Motor 4: Enabled, Power = 50, Gain = F7F8, Offset = FDFC
' Drive Unit 2:
'   MIB I/O Address: 320
'   MIB Mem Address: D0000
'   Motor 1: Enabled, Power = 100, Gain = F8F9, Offset = FDFE
'   Motor 2: Enabled, Power = 100, Gain = 601, Offset = 2
'   Motor 3: Enabled, Power = 100, Gain = C09, Offset = FEFE
'   Motor 4: Enabled, Power = 50, Gain = FA00, Offset = FFC

ROBOT 1 ' XM3064-11-PN, XM3064-11PN
  ARCH 0, 30, 30
  ARCH 1, 40, 40
  ARCH 2, 50, 50
  ARCH 3, 60, 60
  ARCH 4, 70, 70
  ARCH 5, 80, 80
  ARCH 6, 90, 90
  ARMSET 0, 0, 0, 0, 0, 0
' CALPLS: Undefined
  HOFS 0, 0, 0, 0
  HOMESET 0, 0, 0, 0
  HORDR 4, 11, 0, 0
```

```
    ' HTEST: 0, 0, 0, 0
      MCOFS 11, -5943, -8040, 2330, 42620, 0
      MCORDR 12, 3, 0, 0
      RANGE -90112, 90112, -60075, 60075, -92160, 0, -172032,
    172032
      WEIGHT 5, 0
      XYLIM 0, 0, 0, 0
    ROBOT 2 ' EC250-120, EC251S
      ARCH 0, 30, 30
      ARCH 1, 40, 40
      ARCH 2, 50, 50
      ARCH 3, 60, 60
      ARCH 4, 70, 70
      ARCH 5, 80, 80
      ARCH 6, 90, 90
      ARMSET 0, 125, 0, 0, 125, 0
    ' CALPLS: Undefined
      HOFS 0, 0, 0, 0
      HOMESET 0, 0, 0, 0
      HORDR 4, 11, 0, 0
    ' HTEST: 0, 0, 0, 0
      RANGE 0, 163840, -76800, 76800, -36864, 0, -46695, 46695
      WEIGHT 1, 125
      XYLIM 0, 0, 0, 0
>
```

Verwandte Befehle

Verinit

Ver Beispiel

```
> Ver
```

Verinit Befehl

Initialisiert die Robotersystem-Parameter.

Syntax

Verinit

Beschreibung

Verinit stellt Vorgabewerte für alle Roboter ein.

Verinit initialisiert die folgenden Daten

Befehl	Standardwert
Accel	Variiert dem Manipulator-Typ entsprechend
AccelS	Variiert dem Manipulator-Typ entsprechend
Arch	0, 30, 30 1, 40, 40 2, 50, 50 3, 60, 60 4, 70, 70 5, 80, 80 6, 90, 90
Arm	0
ArmSet	Löscht die Nutzer-Definitionen (Arm 1 bis 3). Arm 0 bleibt unverändert.
Base 0	Entspricht dem ursprünglichen Roboter-Koordinatensystem
Base 1-15	Gelöscht
CtrlDev	PC
Fine	Variiert entsprechend dem Manipulator-Typ
HomeSet	Gelöscht
Hordr	Variiert entsprechend dem Manipulator-Typ
LimZ	0
LOCAL 0	Entspricht dem ursprünglichen Roboter-Koordinatensystem
LOCAL 1-15	Gelöscht
MCordr	Variiert dem Manipulator-Typ entsprechend
Speed	Variiert dem Manipulator-Typ entsprechend
SpeedS	Variiert dem Manipulator-Typ entsprechend
TLSet	Löscht die Nutzer-Definitionen (Tool 1 bis 3). Tool 0 bleibt unverändert.
Tool	0
Weight	Variiert dem Manipulator-Typ entsprechend
XYLim	0, 0, 0, 0

Hinweis

Dieser Befehl ist ausschließlich zu Wartungszwecken zu verwenden:

Dieser Befehl ist für Wartungszwecke vorgesehen. **Verinit** initialisiert Werte wie oben beschrieben. In den Fällen, in denen die Werte der Initialisierungsanweisung nicht gewünscht sind, sollten Sie diese nach Ausführung des **Verinit**-Befehls wieder ändern.

Verwandte Befehle

Ver

Beispiel eines Verinit-Befehls

```
> verinit
Initializing parameters for all robots...
>
```

Wait Anweisung

> S

Veranlasst das Programm, entweder eine definierte Zeit lang zu warten, oder solange zu warten, bis die spezifizierte Eingangsbedingung (unter Verwendung der Befehle Sw$ oder Sw) erfüllt ist. (An Stelle von Sw kann auch Oport verwendet werden, um die Hardware-Ausgänge zu überprüfen.)

Syntax

(1) Wait *time*
(2) Wait *inputCondition*
(3) Wait *inputCondition, time*

Parameter

time	Real Ausdruck zwischen 0 und 2.147.483, der den Zeitraum darstellt, der abgewartet werden soll, wenn der Wait-Befehl verwendet wird, um zeitbasiert zu warten. Die Zeit wird in Sekunden angegeben. Das kleinste Inkrement ist .01 Sekunden.
Inputcondition	Diese Bedingung muss den Wert TRUE (-1) oder FALSE (0) zurückgeben. Die folgenden Funktionen und Operatoren können für die inputCondition (Eingangsbedingung) verwendet werden: **Funktionen:** Sw, Sw($), In, In($) **Operatoren:** And, Or, Xor, +, * **Andere:** Klammern, um verschiedenen Operationen und Variablen Prioriät einzuräumen.

Beschreibung

(1) Wait mit Zeitintervall

Als Timer verwendet, veranlasst die **Wait**-Anweisung das Programm, für einen bestimmten Zeitraum anzuhalten und führt dann das Programm weiter aus.

(2) **Wait** mit Eingangsbedingung (ohne Zeitinterval):

Wenn **Wait** als bedingte **Wait**-Verriegelung verwendet wird, veranlasst der Wait-Befehl das Programm zu warten, bis eine bestimmte Bedingung (basierend auf Merkern oder Hardware-Eingängen) erfüllt wurde. Wenn nach TMOut das Zeitintervall abgelaufen ist und die **Wait**-Bedingung noch nicht erfüllt wurde, tritt ein Fehler auf. Außerdem kann der Nutzer mehrere Bedingungen mit einem einzelnen Wait-Befehl überprüfen, indem er den And-, Or- oder Xor-Befehl verwendet. (Mehr über **Wait** erfahren Sie im Beispielabschnitt)

(3) Wait mit Eingangsbedingung und Zeitintervall:

Spezifiziert Wait-Bedingung und Zeitintervall. Wenn die Wait-Bedingung erfüllt ist oder das Zeitintervall abgelaufen ist, springt die Programmsteuerung zum nächsten Befehl. Verwenden Sie Tw, um zu verifizieren, ob die Wait-Bedingung erfüllt wurde, oder ob das Zeitintervall abgelaufen ist.

Wichtige Hinweise

Befehle, die mit Wait arbeiten

Die **Wait**-Anweisung kann mit den Befehlen Sw- oder Sw$ verwendet werden. Sw wird verwendet, um einzelne Hardware-Eingänge zu überprüfen. Sw$ wird verwendet, um Merker zu überprüfen (Siehe „Einschränkungen“ unten).

Die **Wait**-Anweisung kann ferner mit dem Oport-Befehl verwendet werden, um den Status der Hardware-Ausgänge zu überprüfen.

Um 8 I/O-Kanäle gleichzeitig zu überprüfen, kann der **Wait**-Befehl mit den Befehlen In oder In$ verwendet werden. In wird verwendet, um 8 Hardware-Eingänge gleichzeitig zu überpüfen. In$ wird verwendet, um 8 Merker gleichzeitig zu überprüfen.

Der Unterschied zwischen Wait (Sw) und Wait Sw($)

Für den Nutzer ist es sehr wichtig, den Unterschied zwischen den Befehlen Sw und Sw $ zu verstehen, wenn diese mit dem **Wait**-Befehl verwendet werden. Der Wait-Befehl erlaubt es dem System, basierend entweder auf Hardware-Eingängen (unter Verwendung des Sw-Befehls mit Wait) oder Merkern (unter Verwendung des Sw $-Befehls mit **Wait**) zu warten. Der Sw **$** -Befehl arbeitet mit internen Merkern (normalerweise für Status-Zwecke verwendet) und hat keinerlei Auswirkungen auf die Hardware I/O des Roboters. Der Sw-Befehl arbeitet mit den Hardware Eingangsports. Diese Hardwareports sind einzelne Eingänge, die mit Geräten außerhalb der Steuerung kommunizieren.

Spezifikation einer Zeitüberschreitung (Timeout) für den Gebrauch mit Wait

Wenn der **Wait**-Befehl ohne ein Zeitintervall verwendet wird, kann eine Zeitüberschreitung spezifiziert werden. Diese setzt ein Zeitlimit für das Warten auf den definierten Sw- oder Sw $-Befehl. Diese Zeitüberschreitung wird durch den TMOut-Befehl gesetzt. Bitte lesen Sie die Beschreibung dieses Befehls für weitere Informationen. (Die Vorgabeeinstellung für TMOut ist 0. Das bedeutet, dass keine Zeitüberschreitung definiert ist.)

Verwandte Befehle

In, In $, InBCD, Off, Off $, On, On $, Op, Oport, Out, Out $, OutW, Sw, Sw $, TMOut

Wait Beispiel

Das folgende Beispiel zeigt 2 Tasks. Jeder der beiden Tasks hat die Möglichkeit, Bewegungsbefehle zu initiieren. Jedoch wird ein Sicherungsmechanismus zwischen den beiden Tasks verwendet, um sicherzustellen, dass ein Task erst dann die Kontrolle über die Bewegungsbefehle des Roboters erhält, wenn der andere Task deren Verwendung abgeschlossen hat. Dies ermöglicht es 2 Tasks, Bewegungsbefehle korrekt und in geordneter, vorhersehbarere Art und Weise, auszuführen. Sw $ wird in Kombination mit dem Wait-Befehl verwendet, um zu warten bis der Merker Nr. 1 den richtigen Wert erreicht hat, von dem an es sicher ist, eine neue Bewegung auszuführen.

```
Function main
    Integer I
    Off $1
    Xqt !2, task2
    For I = 1 to 100
       Wait Sw($1) = Off
       Go P(i)
       On $1
    Next I
Fend

Function task2
    Integer I
    For I = 101 to 200
       Wait Sw($1) = On
       Go P(i)
       Off $1
    Next I
Fend


Wait Sw(0) = 1  ' Wartet, bis der Hardware-Eingang #0 1 entspricht

Wait 60.5       ' Wartet 60.5 Sekunden und setzt dann die Ausführung
fort

Wait Sw(0) = 0 And Sw(1) = 1  ' Wartet, bis beide Bedingungen
                               ' Sw(0)=0) und Sw(1)=1)
                               ' erfüllt sind

Wait Sw($1) = 0 Or Sw(1) = 1  ' Wartet, bis beide Bedingungen
                               '(Sw($1)=1) und Sw(1)=1
                               ' erfüllt sind
Wait 1;On 1  'Wartet eine Sekunden und schaltet dann den Ausgang 1 ein.
```

WaitNet Anweisung

S

Wartet darauf, dass die Netzwerkverkbindung hergestellt wird.

Syntax
WaitNet #*portNumber* [, *timeOut*]
Parameter

portNumber	Integer Ausdruck für die zu öffnende Schnittstellennummer. Der Bereich liegt zwischen 128 und 131.
timeOut	Optional. Maximale Zeit, die für die Verbindung abgewartet werden soll.

Verwandte Befehle
Wait, WaitSig, WaitPos

Beispiel einer WaitNet Anweisung

In diesem Beispiel sind die TCP/IP-Einstellungen von zwei PC's wie folgt konfiguriert:

PC #1:
Port: #128
Host Name: PC2
TCP/IP Port: 1000

```
Function tcpip1
  OpenNet #128
  WaitNet #128
  Print #128, "Data from host 1"
Fend
```

PC #2:
Port: #128
Host Name: PC1
TCP/IP Port: 1000

```
Function tcpip2
  String data$
  OpenNet #128
  WaitNet #128
  Input #128, data$
  Print "received '", data$, "' from host 1"
Fend
```

WaitPos Anweisung

S

Wartet bis der Roboter im CP-Modus zum Stillstand verzögert hat, bevor die nächste Anweisung ausgeführt wird.

Syntax

WaitPos [*robotNumber*]

Parameter

robotNumber Optional. Spezifiziert, auf welchen Roboter gewartet werden soll.

Verwandte Befehle

Wait, WaitSig, Cp

Beispiel einer WaitPos Anweisung

```
Off 1
CP On
Move P1
Move P2
WaitPos  ' Wartezeit, bis der Roboter verzögert
On 1
CP Off
```

WaitSig Anweisung

S

Wartet auf ein Signal von einem anderen Task.

Syntax

WaitSig *signalNumber* [, *timeOut*]

Parameter

signalNumber	Integer Ausdruck, der für die zu empfangende Signalnummer steht. Der Bereich liegt zwischen 0 und 127.
timeOut	Optional. Real Ausdruck, der die maximale Wartezeit repräsentiert.

Beschreibung

Verwenden Sie **WaitSig**, um auf ein Signal eines anderen Tasks zu warten. Das Signal wird erst empfangen, nachdem **WaitSig** begonnen hat. Vorangegangene Signale werden ignoriert.

Verwandte Befehle

Wait, WaitPos, Signal

WaitSig Beispiel

```
Function Main
  Xqt SubTask
  Wait 1
  Signal 1
  .
  .
Fend

Function SubTask
  WaitSig 1
  Print "signal received"
  .
Fend
```

Weight Anweisung

> S

Definiert die Last des Roboterarms bzw. gibt es aus.

Syntax

Weight [*payloadWeight* [*, distance*]]

Parameter

payloadWeight	Das zu tragende Gewicht in kg.
distance	Der Abstand zwischen dem Rotationsmittelpunkt des zweiten Arms zum Schwerpunkt des Greifers, angegeben in mm.

Rückgabewerte

Zeigt die aktuellen **Weight** Einstellungen an, wenn die Parameter weggelassen werden.

Beschreibung

Spezifiziert Parameter zur Kalkulation der maximalen Beschleunigung von PTP-Bewegungen. Die **Weight**-Anweisung spezifiziert das Gewicht des Greifers und der zu tragenden Teile.

Die Spezifikation der Armlänge (*distance*) ist nur bei SCARA-Robotern notwendig. Die Armlänge ist der Abstand von der zweiten Arm-Rotationsachsen-Mittellinie zum kombinierten Schwerpunkt aus Roboterhand und Arbeitsstück.

Wenn der Äquivalenzwert des Werkstück-Gewichtes, welcher aus den spezifizierten Parametern berechnet wird, die maximal erlaubte Nutzlast überschreitet, tritt ein Fehler auf.

Potentielle Fehler

Das Gewicht überschreitet das Maximum.

Wenn die äquivalente Last, welche aus dem eingegebenen Wert kalkuliert wird, die maximale Last überschreitet, tritt ein Fehler auf.

Potentieller Schaden des Manipulatorarms

Beachten Sie, dass die Spezifikation eines **Weight** Roboterhandgewichts, welches bedeutsam niedriger ist als das Gewicht des tatsächlichen Werkstückes, überhöhte Beschleunigung und Verzögerung zur Folge haben kann. Dies wiederum kann kann schwere Schäden am Manipulatorarm verursachen.

Hinweis

Weight-Werte werden durch Abschalten der Versorgungsspannung nicht geändert.

Die **Weight**-Werte werden durch Abschalten der Stromzufuhr nicht geändert. Die Werte werden jedoch initalisiert, wenn der Verinit-Befehl ausgeführt wird. Die Ausgangswerte variieren je nach Typ des Manipulatorarms. Ziehen Sie das Manipulatorarm-Handbuch zu Rate, um mehr über die **Weight**-Augangswerte zu erfahren.

Verwandte Befehle

Accel, Verinit

Beispiel einer Weight-Anweisung

Dieser Weight-Befehl im Online-Fenster zeigt die aktuelle Einstellung an.

```
> weight
2.000, 200.000
>
```

Weight Funktion

F

Gibt einen Weight-Parameter aus.

Syntax

Weight(*paramNumber*)

Parameter

paramNumber Integer Ausdruck, der einen der unten angegebenen Werte enthält:

1: Nutzlast-Gewicht

2: Armlänge

Rückgabewerte

Real Zahl, welche den Parameterwert beinhaltet.

Verwandte Befehle

Beispiel einer Weight-Funktion

```
Print "The current Weight parameters are: ", Weight(1)
```

While... Wend

S

Führt die angegebenen Anweisungen aus, solange die spezifizierte Bedingung erfüllt ist.

Syntax

While [*condition*]
[*statements*]
Wend

Parameter

condition	Eine beliebige gültige Bedingung, welche die Überprüfung der E/A mit dem Sw-Befehl oder der Vergleich von Variablen sein kann.
statements	Optional. Eine oder mehrere Anweisungen werden wiederholt, während die *condition (Bedingung)* WAHR ist.

Beschreibung

Spezifiziert die While-Bedingung. Wenn erfüllt, werden die Anweisungen zwischen While und Wend ausgeführt. Dann wird die While-Bedingung erneut geprüft. Die Ausführung der While... Wend-Anweisungen und das Nachprüfen der While-Bedingung dauert an, solange die While-Bedingung erfüllt ist.

Wenn die While-Bedingung nicht erfüllt wird, springt die Programmsteuerung zu dem Befehl, der auf Wend folgt. Für eine While-Bedinung, die beim ersten Prüfen nicht erfüllt wird, werden die Anweisungen innerhalb der While...Wend-Anweisung niemals ausgeführt.

Hinweise

Regeln für While...Wend:

- Auf jedes **While** muss ein **Wend** folgen.
- Innerhalb einer **While…Wend**-Schleife können bis zu 16 **While…Wend**-Schleifen verschachtelt sein. Jedes **Wend** entspricht dem vorausgehenden **While**. Ein **Wend** ohne vorausgehendes **While** führt zu einem Fehler.
- In der **While**-Bedingung kann jeder gültig Operator enthalten sein.

Verwandte Befehle

If/Then/Else

While-Beispiel

```
Long i

i = 1
While i < 60    'Führt Anweisungen zwischen While/Wend aus, wenn i<60
 .
 .
   i = i + 2
Wend
```

WOpen Anweisung

S

Öffnet eine Datei zum Schreiben.

Syntax

WOpen *fileName* **As** #*fileNumber*

Parameter

fileName	Ein Zeichenkettenausdruck, der den Namen der Datei enthält, aus der gelesen werden soll. Laufwerk und Pfad können ebenfalls enthalten sein.
fileNumber	Integer zwischen 30 und 63, welche als Bezeichner für die Datei verwendet wird.

Beschreibung

Öffnet den spezifizierten Dateinamen zum Schreiben und identifiziert ihn durch die spezifizierte *fileNumber*. Diese Anweisung wird verwendet, um eine spezifizierte Datei zu öffnen und Daten in diese Datei zu schreiben. Der Befehl **Close** schließt die Datei und gibt die Dateinummer aus. (Um Daten anzuhängen, lesen Sie bitte die Erklärung zu Aopen.)

Wenn der spezifizierte Dateiname nicht auf dem aktuellen Verzeichnis der Diskette existiert, erstellt die Anweisung **WOpen** die Datei und beschreibt sie. Wenn der spezifizierte Dateiname existiert, löscht Wopen alle Daten in dieser Datei und beschreibt sie neu.

Die *fileNumber* identifiziert die Datei so lange, wie die Datei geöffnet ist. Die Datei wird von der Input#-Anweisung zum Lesen, von der Print#-Anweisung zum Drucken und zum Schließen (Close#) verwendet. Entsprechend kann die Dateinummer nicht zur Spezifikation einer anderen Datei verwendet werden, bis die aktuelle Datei geschlossen ist.

Verwandte Befehle

AOpen, Close, Input #, ROpen

WOpen-Beispiel

Das unten gezeigte, einfache Beispiel öffnet eine Datei, schreibt einige Daten hinein, öffnet dieselbe Datei später wieder und liest deren Daten in eine Feldvariable.

```
Real data(100)

For I = 0 To 100
    data(i) = i
Next I

WOpen "TEST.VAL" As #30
For I = 0 To 100
    Print #30, data(i)
Next I
Close #30

ROpen "TEST.VAL" As #30
For I = 0 to 100
    Input #30, data(I)
Next I
Close #30
```

Write Anweisung

S

Schreibt Zeichen in eine Datei ohne Endzeichen am Zeilenende.

Syntax

Write #*fileNumber*, *string*

Parameter

fileNumber	Zu beschreibende Datei oder Kommunikationsschnittstelle.
string	Zeichenkettenausdruck, der in die Datei geschrieben wird.

Beschreibung

Der Unterschied zwischen Write und Print besteht darin, dass bei Write am Zeilenende kein Endzeichen gesetzt wird.

Der aktuelle Dateizeiger wird mit der Länge des Zeichenkettenausdrucks vorgeschoben.

Verwandte Befehle

Print, Read, Seek, WOpen

Write-Beispiel

```
WOpen "test.dat" As #30
For i = 1 to 10
    Write #30, data$(i)
Next i
Close #30
```

Xor Operator

> S

Führt die bitweise Xor-Operation (EXKLUSIV-ODER-Verknüpfung) zwischen den Werten der Operanden aus.

Syntax

Operand **Xor** *Operand*

Parameter

Operand Ein numerischer Wert oder ein Variablenname.

Rückgabewerte

Bitweise Xor-verknüpfter Wert der Operanden.

Beschreibung

Der Xor-Operator führt die bitweise Xor-Operation (EXKLUSIV-ODER-Verknüpfung) zwischen den Werten der Operanden aus. Jedes Bit des Ergebnisses ist der Xor-verknüpfte Wert zwei einander entsprechender Bits der beiden Operanden.

Verwandte Befehle

And, LShift, Not, Or, RShift

Xor-Operator Beispiel

```
>print 2 Xor 6
 4
>
```

Xqt Anweisung

S

Beginnt die Ausführung eines Tasks aus einem anderen Task heraus.

Syntax

Xqt [*taskNumber,*] *funcName* [**(***argList***)**] [, **NoPause**]

Parameter

taskNumber	Optional. Die Tasknummer des auszuführenden Tasks. Der Bereich für die Tasknummer liegt zwischen 1 und 32. **Hinweis:** Die SPEL-Sprache der SRC-3xx-Serie-Steuerungen hat ein Ausrufezeichen vor der Tasknummer verwendet. Das Ausrufezeichen ist nicht mehr notwendig und wird ignoriert.
funcName	Der Name der auszuführenden Funktion.
argList	Optional. Liste von Argumenten, die an die Funktionsprozedur weitergegeben werden, wenn diese aufgerufen wird. Multiple Argumente werden durch Kommata voneinander getrennt.
NoPause	Optional. Spezifiziert, dass der Taks nicht pausiert, wenn eine Pauseanweisung oder ein Signal eintritt.

Beschreibung

Xqt startet die spezifizierte Funktion und kehrt unverzüglich zurück.

Normalerweise wird der Parameter *taskNumber* nicht benötigt. Wenn *taskNumber* weggelassen wird, ordnet SPEL der Funktion automatisch eine Tasknummer zu, so müssen Sie nicht die Übersicht darüber behalten, welche Tasknummern gerade verwendet werden.
Hinweise

Die SPEL-Sprache der SRC-3xx-Serie-Kontroller hat *startLine*- und *endLine*-Argumente unterstützt . Diese werden in SPEL+ nicht unterstützt.

Verwandte Befehle

Chain, Function/Fend, Halt, Resume, Quit

Xqt Beispiel

```
Function main
    Xqt flash, NoPause 'Startet die Funktion Flash als Task 2
    Xqt Cycle(5)       'Startet die Funktion Cycle als Task 3

    Do
        Wait 3           'Führt den Task 2 drei Sekunden lang aus
        Halt flash     'Unterbricht den Task

        Wait 3
        Resume flash   'Setzt den Task fort
    Loop
Fend

Function Cycle(count As Integer)
    Integer i

    For i = 1 To count
        Jump pick
        On vac
        Wait .2
        Jump place
        Off vac
        Wait .2
    Next i
Fend

Function flash
    Do
        On 1
        Wait 0.2
        Off 1
        Wait 0.2
    Loop
Fend
```

XY Funktion

F

Gibt einen Punkt von individuellen Koordinaten aus, die in einem Punktausdruck verwendet werden können.

Syntax

XY(*x, y, z, u***)**

Parameter

x Real Ausdruck, der die X-Koordinate repräsentiert.
y Real Ausdruck, der die Y-Koordinate repräsentiert.
z Real Ausdruck, der die Z-Koordinate repräsentiert.
u Real Ausdruck, der die U-Koordinate repräsentiert.

Rückgabewerte

Ein Punkt, der auf der Basis der definierten Koordinaten konstruiert wird.

Verwandte Befehle

JA, Point Expression (Punktausdruck)

Beispiel einer XY-Funktion

```
P10 = XY(60, 30, -50, 45) + P20
```

XYLim Anweisung

> S

Stellt die erlaubten Grenzwerte des Arbeitsbereichs der XY-Ebene für den Roboter ein oder zeigt diese an.

Syntax

XYLim [*minX, maxX, minY, maxY*]

Parameter

minX Die minimale X-Koordinatenposition, zu welcher der Manipulatorarm verfahren kann. (Der Manipulatorarm darf zu keiner Position verfahren, deren X-Koordinate kleiner als *minX* ist.)

maxX Die maximale X-Koordinatenposition, zu welcher der Manipulatorarm verfahren kann. (Der Manipulatorarm darf zu keiner Position verfahren, deren X-Koordinate größer als *maxX* ist.)

minY Die minimale Y-Koordinatenposition, zu welcher der Manipulatorarm verfahren kann. (Der Manipulatorarm darf zu keiner Position verfahren, deren Y-Koordinate kleiner als *minY* ist.)

maxY Die maximale Y-Koordinatenposition, zu welcher der Manipulatorarm verfahren kann. (Der Manipulatorarm darf zu keiner Position verfahren, deren Y-Koordinate größer als *maxY* ist.)

Rückgabewerte

Zeigt die aktuellen **XYLim**-Werte, wenn keine Parameter verwendet werden.

Beschreibung

XYLim wird verwendet, um die Arbeitsbereich-Grenzwerte der XY-Ebene zu definieren. Viele Robotersysteme erlauben es dem Anwender, Arbeitsbereichsgrenzen zu definieren, die SPEL-Sprache jedoch gestattet es, sowohl die Arbeitsbereichsgrenzen als auch die Grenzwerte für die XY-Ebene zu definieren. In der Tat ist es den Nutzern so möglich, einen XY-Ebenen-Arbeitsbereich für ihre Anwendung anzulegen. (Bedenken Sie, dass die Grenzwerte des Achsenbereichs ebenfalls mit SPEL definiert werden können.)

Der Arbeitsbereich, der mit **XYLim**-Werten angelegt wurde, bezieht sich nur auf die Zielpositionen von Bewegungsbefehlen und nicht auf Bewegungsbahnen von der Start- zur Zielposition. Daher kann sich der Arm während der Bewegung außerhalb des XYLim-Bereichs bewegen. (D.h. **der XYLim**-Bereich hat keine Auswirkungen auf den Pulse-Befehl.)

Hinweise

Ausschalten der Überprüfung der XY-Ebenen-Begrenzung

Es gibt viele Anwendungen, für die keine Überprüfung der XY-Ebenen-Begrenzung nötig ist. Aus diesem Grund gibt es eine einfache Methode, diese Überprüfung auszuschalten. Um die Überprüfung der XY-Ebenen-Begrenzung auszuschalten, setzen Sie die XY-Ebenen-Begrenzungswerte *minX*, *maxX*, *minY* und *maxY* auf 0. Zum Beispiel XYLim 0, 0, 0, 0.

Vorgabe-Grenzwerte der XY-Ebene

Es gibt einige Fälle, die das System dazu veranlassen, alles auf die ursprünglichen Vorgabewerte zurückzusetzen. Zum Beispiel veranlasst der Verinit-Befehl, dass alle Systemwerte auf ihre Vorgabewerte zurückgesetzt werden. Die Vorgabewerte für den XYLim-Befehl lauten „0, 0, 0, 0“. (Überprüfung der XY-Ebenen-Begrenzung ist ausgeschaltet.)

Tipp

Point & Click-Installation für XYLim

EPSON RC+ hat ein “Point & Click”-Dialogfenster zur Definition der Grenzwerte der XY-Ebene. Die einfachste Methode, um die XYLim-Werte zu setzen, ist die Verwendung des XYLim-Karteikarte: Roboterparameter-Befehl (Projektmenü).

Verwandte Befehle

XYLim Karteikarte: Roboterparameter-Befehl (Projektmenü)
Range

Beispiel einer XYLim Anweisung

Dieses einfache Beispiel vom Online-Fenster stellt die aktuellen XYLim-Einstellungen ein und zeigt sie dann an.

```
> xylim -200, 300, 0, 500

> XYLim
-200.000, 300.000, 0.000, 500.000
```

XYLim Funktion

F

Gibt Punktdaten für die obere oder die untere Grenze des XYLim-Bereiches aus.

Syntax

XYLim(*limit*)

Parameter

limit	Integer Ausdruck, der spezifiziert, welche Grenze ausgegeben werden soll. 1: Untergrenze. 2: Obergrenze.

Rückgabewerte

Punkt, der die spezifizierten Grenzkoordinaten beinhaltet.

Verwandte Befehle

XYLim Anweisung

Biespiel einer XYLim Funktion

```
P1 = XYLim(1)
P2 = XYLim(2)
```

ZeroFlg Funktion

F

Gibt den Status des Merkers vor dem letzen Ein- oder Ausschalten aus.

Syntax

ZeroFlg

Rückgabewerte

Gibt den Status des Merkers vor dem letzen Ein- oder Ausschalten aus. (Gibt eine 0 oder eine 1 aus.)

Beschreibung

ZeroFlg dient der exklusiven Steuerung einer einzelnen Ressource (wie z.B. einem RS232-Port) während mehrere Tasks gleichzeitig laufen.

Nach dem Einschalten eines Merkers verwenden Sie **ZeroFlg** um festzustellen, ob der aktuelle Task der Task ist, der den Merker eingeschaltet hat (wohingegen einige andere Tasks den Merker zur selben Zeit einschalten). Wenn der aktuelle Task den Merker eingeschaltet hat, gibt **ZeroFlg** 0 aus, und zeigt damit an, dass der vorangegangene Wert 0 war.

Sie müssen ZeroFlg umgehend nach Einschalten des Merkers verwenden.

Zum Beispiel:

```
On $1
If ZeroFlg = 0 Then
  Print "This task turned on $1"
Else
  Print "Some other task turned on $1"
EndIf

See Also
Off $, On $
ZeroFlg Example

Dieses Beispiel verwendet zwei Funktionen, um zu einer Zeit
ausschließlich einem Task zu erlauben, mit dem Gerät zu kommunizieren,
welches mit dem RS232 Port 1 verbunden ist. Die Funktionen lauten
AccessPort und ReleasePort. Wenn ein Task den Port verwendet, wartet der
andere Task solange, bis er die Kontrolle über den Port erhält.

Function main

  ' Initialisiert das Access Flag
  Off $port

  Xqt task2

  Do
    AccessPort
    Print #1, "main"
    ReleasePort
  Loop
Fend

Function task2

  Do
    AccessPort
    For i = 0 To 100
      Print #1, i
    Next
    ReleasePort
  Loop
Fend

Function AccessPort
```

```
Do
  ' Schaltet das Access Flag ein
  On $port
  ' Überprüft, ob der Task eingeschaltet ist
  If ZeroFlg = 0 Then
    Dieser Task hat die Kontrolle, also wird die Funktion verlassen
    Exit Function
  EndIf
  ' Wartet auf den Steuertask,
  ' um die Ressource freizugeben
  Wait Sw($port) = Off
Loop
Fend

Function ReleasePort

  Off $port
  ' Gewährte Zeit für einen anderen Task,
  ' um Zugriff zu erhalten
  Wait .01
Fend
```

SPEL+ Fehlermeldungen

Um Hilfe zu irgendeinem SPEL^{+}-Fehler zu erhalten, platzieren Sie den Cursor auf der Fehlermeldung, die auf dem Online-Fenster angezeigt wird, und drücken Sie die Taste F1.

Error #	Message
2	Syntax-Fehler oder undefinierte Angaben.
3	RETURN wurde ohne GOSUB ausgeführt.
4	Zu viele GOTO, GOSUB oder Sprungmarken. (Maximum 512)
5	Numerischer Wert außerhalb des gültigen Bereichs.
6	Überlauf von numerischen Werten oder Variablen.
7	Es gibt mehr verschachtelte GOSUB-Anweisungen als erlaubt [16].
9	Feldvariable außerhalb der Definition.
16	Eine Programmzeile befindet sich nicht innerhalb von FUNCTION ... FEND.
19	Zu viele Zeichen in einer Zeile.
27	Angewählter E/A existiert nicht.
30	INPUT: Anzahl der Daten / Variablen ungleich.
33	Pufferspeicher-Überlauf von RS-232C
34	Paritäts-/Überlauf-/Anordnungsfehler von RS-232C
36	Zu viele Zeichen an den RS-232C-Port gesendet.
37	Overtime-Fehler der RS232-Kommunikation.
53	Datei existiert nicht.
57	Es existiert bereits eine Datei mit diesem Namen.
58	Der Dateiname ist falsch oder existiert bereits.
62	Falscher Dateiname / Datei kann nicht geöffnet werden.
63	Diskette nicht bereit.
65	Disketten-Lesefehler.
66	Disketten-Schreibfehler.
77	Punktdefinition nicht korrekt.
78	Undefinierter Punkt.
88	E/A ist bereits eine Remotefunktion.
92	Unterschiedliche Eingangssignale im Türsicherheitskreis.
94	Wartezeit für WAIT SW ist überschritten.
119	Arm bewegt sich nach dem Ausschalten zu viel.
120	Gebrauch eines besonderen Befehls in nicht korrekter Einstellung
121	Befehl während NOT-AUS nicht ausführbar.
122	Unzulässiger Datentyp wird verwendet.
123	Der Befehl wird vom Steuergerät nicht unterstützt.
124	Nummerischer Wert Bereichsüberschreitung.
125	Arm erreicht Grenzen des Arbeitsbereichs.
126	Arm erreicht XYLIM Grenzen.
129	LIMZ Fehler.
143	HOME Position nicht definiert.
146	Bewegung unter SFREE nicht möglich.
150	Befehl unter MOTOR OFF nicht ausführbar.
151	Position innerhalb FINE nicht erreichbar.

Error #	Message
173	Max. Drehmoment im Low-Power-Modus überschritten.
174	Max. Drehmoment im High-Power-Modus überschritten.
185	Unterbrechung Encodersignal.
190	Encoder ist überhitzt.
191	Encoderüberdrehzahl.
193	Batteriefehler im Manipulatorsockel.
194	Encoder Prüfsummenfehler.
195	Encoder Back-up Alarm.
230	Der Befehl MCORG wurde nicht ausgeführt.
231	Der Befehl MCAL wurde nicht ausgeführt.
233	Fehler bei der Erfassung des Encoder-Z-Phasen-Signals.
359	Die Option Barcode-lesen ist nicht freigeschaltet.
360	Undefinierte Vision-Sequenz.
361	Undefiniertes Vision-Objekt.
362	Undefinierte Vision-Eigenschaft.
363	Vision-Objekt nicht gefunden.
364	Ungültiger Wert für die Kalibrier-Einstellung.
365	Die vorhergehende Sequenz hat keine Zuordnung.
366	Ungültiger Eigenschaftswert.
367	Ungültiges Ergebnis für dieses Objekt.
368	Ungültige Eigenschaft oder Ergebnis.
369	Falsche Anzahl von Argumenten.
370	Modell wurde nicht eingelernt.
371	Kalibrierung nicht vollständig.
372	Undefinierte Kalibrierung.
373	Kalibrierungspunkte wurden nicht eingerichtet.
374	Drucker-Fehler.
375	Keine Steuerung.
376	Falscher Objekttyp.
377	Vision-System-Fehler.
378	Sequenz existiert bereits.
379	Pfad wurde nicht gefunden.
380	Ungültige Kamera.
381	Falsches String-Format.
382	Ungültige Bildatei.
383	Die Eigenschaft "angle enabled" wurde nicht aktiviert.
384	Objekt existiert bereits.
385	Markierung für die Kalibrierung nicht gefunden.
386	Der Wert der Kalibrier-Sequenz muß für "Number to find" gleich 9 sein.
387	Nicht genügend Schritte in der Kalibrier-Sequenz.
388	Für Kamera unterhalb des Roboter-Arbeitsbereichs wurde keine Sequenz festgelegt.
389	Kalibrier-Sequenz für Kamera unterhalb des Roboter-Arbeitsbereichs ist nicht vorhanden.
390	Kalibrier-Sequenz für Kamera unterhalb des Roboter-Arbeitsbereichs ist nicht vollständig.
391	Die Referenz-Kalibrierung ist unzureichend.
392	Keine Kalibrier-Sequenz ausgewählt.
393	Ungültige String-länge für die Kalibrierung oder Prüfung.
394	Ungültiger Zeichensatz.
395	Ungültiger Kalibrierungs-String-Wert.
396	Ungültiger "Character-Position"-Parameter für "Constraints".
397	Ungültige CalString länge.
398	OCR-Kalibrierung fehlerhaft.
399	Keine Objekte in dieser Sequenz.
400	FEND ohne Function gefunden.
401	Function ohne FEND gefunden.

Error #	Message
402	Doppelte Zeilennummer.
403	Falsche Zeilennummer.
404	Zeilennummer zu groß.
405	Funktion existiert bereits als DLL Funktion.
407	Task wurde durch NOT-AUS beendet.
408	Punktedatei für den aktuellen Roboter nicht gefunden.
409	Ungültiger Befehl für den aktuellen Roboter.
410	Zeitüberschreitung bei Speicherzugriff.
411	Ungleiche Typen.
412	Zeitüberschreitung bei DeviceNet-Zugriff.
413	Falsche Größe zugeteilt.
414	Es sind keine Gruppen für das aktuelle Projekt eingerichtet.
415	Speicherüberlauf.
416	Kein Bereich.
417	SPEL OLE Kanal ist belegt.
418	Zeitüberlauf bei einem SPEL OLE Befehl.
419	Zeitüberlauf bei Zugriff auf SPEL+.
420	Zeitüberlauf bei Zugriff auf Ethernet E/A.
421	Abfangroutine läuft bereits.
422	SPWLIB30 OLE Fehler.
423	Task wurde abgebrochen.
424	HP-Status (Power High) wurde durch LP-(Power Low)-Remote-Eingang ausgeschaltet.
425	COM Schnittstelle wird nicht unterstüzt.
426	Kann bei geöffneter Schicherheitsabschrankung nicht fortgesetzt werden.
500	Der Befehl "ALL" wird erwartet.
501	Der Befehl "AS" wird erwartet.
502	Der Befehl "=" wird erwartet.
503	Ungültige Armnummer, Werte von 0 bis 4 sind gültig.
504	Ungültiges Punktemerkmal.
505	Ungültige Achsangabe, Werte X, Y, Z, oder U sind gültig.
506	Ungültige Dateinummer.
507	Ungültige Bezeichnung.
508	Bei Include-Dateien sind nur #-Anweisungen gültig.
509	Ungültige Palettennummer, Werte von 0 bis 15 sind gültig.
510	Ungültiger Punkt.
511	Ungültige Feldvariable.
512	Ungültige Toolnummer, Werte von 0 bis 4 sind gültig.
513	Ungültige Übertragungsrate.
514	Der Befehl "CALL" wird erwartet.
515	Der Befehl "CALL","GOTO",oder "GOSUB" wird erwartet.
516	Es wird ein Doppelpunkt erwartet.
517	Es wird ein Komma erwartet.
518	Ungültige Datenbits.
519	Ein Datentyp wird erwartet.
520	Ungültige Deklaration innerhalb einer Funktion.
521	" #define " kann innerhalb einer Funktion nicht verwendet werden.
522	Zu viele Dimensionen.
523	Erwartet einen DLL-Alias.
524	Eine linke Klammer oder "AS" wird erwartet.
525	Erwartet einen DLL-Pfad und einen Dateinamen.
526	Doppelter Funktionsname.
527	Der Befehl "ELSE" wird erwartet.
528	Ein Aufrufezeichen wird erwartet.
529	Der Befehl "Do", "For", oder "Function" wird erwartet.
530	Eine Punktenummer muss angegeben werden.
531	Eine Nummer für die Abfangroutine oder ein Modus wird erwartet.

Error #	Message
532	Ausdruck erwartet.
533	Ein ARRAY wird erwartet .
534	Ungültige Dateinummer. Werte von 30 bis 63 sind gültig.
535	Es wird eine Zahl erwartet.
536	Muss als Funktion verwendet werden.
537	Fehlender Parameter bei SetCom#1, "RTS" oder "None" fehlt.
538	Dateiname mit der Erweiterung .INC erwartet.
539	INPUT erwartet.
540	Ein "Integer" Ausdruck wird erwartet.
541	Ein Schlüsselwort kann nicht als Bezeichnung verwendet werden.
542	Ein Schlüsselwort kann nicht als Bezeichnung verwendet werden.
543	Der Befehl "ON" oder "OFF" wird erwartet.
544	HIGH oder LOW wird erwartet.
545	Darf nicht innerhalb einer Funktion verwendet werden.
546	In Include-Dateien dürfen keine Zeilennummern enthalten sein.
547	Linke Klammer erwartet.
548	Eine Zeilennummer oder ein Label wird erwartet.
549	Der Befehl "MERGE" wird erwartet.
550	Fehlendes Punktattribut.
551	Fehlende Achse.
552	Fehlende Bezeichnung.
553	Eine Zeichenkette wird erwartet.
554	Fehlende Armnummer.
555	Ein Operand wird erwartet.
556	Fehlende Palettennummer.
557	Fehlende Toolnummer.
558	Ausführen von MKVER (Systemdatensicherung).
559	Der Befehl "NEXT" wird erwartet.
560	Es werden keine Argumente erwartet.
561	End of-Anweisung erwartet.
562	Ein Schlüsselwort kann nicht als eine Funktion verwendet werden.
563	Dieser Befehl kann nicht vom Online-Fenster aus ausgeführt werden.
564	Es wird nur eine Variable erwartet, kein Array.
565	Dieser Befehl wird von SPEL nicht unterstützt.
566	Dieser Befehl wird noch nicht unterstützt.
567	Vision System ist nicht aktiviert.
568	Ungültiger Wert für Parallelbearbeitung.
569	Ungültige Anweisung für die Parallelbearbeitung.
570	Anweisung für die Parallelbearbeitung erwartet.
571	Das Argument für die Parallelbearbeitung wird noch nicht unterstützt.
572	Ungültige Parität.
573	Ein Punkt oder Komma wird erwartet.
574	Ein Punkt muß angegeben werden.
575	# erwartet.
576	Ausrufezeichen werden erwartet.
577	Not-Aus, Ausgang oder Roboter muß angegeben werden.
578	Falsche Dimensionsanzahl.
579	Rechte Klammer erwartet.
580	Ein Semikolon wird erwartet.
581	Ausführen von SETVER (Systemdaten zurückspielen).
582	Zeichenkette erwartet.
583	STEP erwartet.
584	Ungültige Stop-bits.
585	Fehlender Parameter bei SetCom#1, "XON" oder "None" fehlt.
586	Tasknummer 1 bis 32 erwartet.
587	Ungültiges Endzeichen.
588	Der Befehl "TO" wird erwartet.

Error #	Message
589	Der Befehle "THEN" wird erwartet.
590	Der Befehl "WHILE" oder "UNTIL" wird erwartet.
591	Nicht definiertes Label.
592	Falsche Anzahl von Argumenten.
593	Keine entsprechende SEND-Anweisung.
594	Ungültige TRAP-nummer, Werte von 1 bis 4 sind gültig.
595	Include-Datei ist nicht vorhanden.
596	Die Include-Datei ist nicht im aktuellen Projekt vorhanden.
597	Ungültiger Include-Dateiname.
598	Die Variable ist bereits als Globale Variable deklariert.
599	Die Variable ist bereits im Modullevel deklariert.
600	Die Variable ist bereits als Funktionsparameter deklariert.
601	Ungleiche Typen.
602	Eine Datei kann nicht sich selbst beinhalten.
603	Eine Punktnummer oder ein "*" wird erwartet.
604	Dateinummer oder Variable erwartet.
605	Zu viele Parameter.
606	Die Include-Datei ist bereits eingebunden.
607	Task Typ erwartet.
608	Ungültiger Task Typ.
609	Kann den Bereich nicht auf eine Variable übertragen. Verwenden Sie BYREF..
610	Dieser Befehl ist nur im Online-Fenster möglich.
611	Task läuft bereits.
612	Ungültiger Parameter.
613	Rechte Klammer wird erwartet.
614	Zu viele ELSE-Anweisungen im Ausdruck.
615	Ein Punkte-Label wird erwartet.
616	Ungültige Timernummer.
617	Durch Ändern des Remote-LP(Power Low)-Eingangs wurden alle Tasks abgebrochen.
618	Der Befehl "GOTO" wird erwartet.
619	SPEL+ Runtime-Treiber nicht aktiviert, Bitte aktivieren Sie den Treiber unter Setup/Optionen.
620	SPEL+ muss neu gestartet werden, bitte überprüfen Sie die Spannungsversorgung zur Drive-Unit.
621	Der Neustart von SPEL+ ist fehlgeschlagen. Bitte überprüfen Sie die Spannungsversorgung zur Drive-Unit..
700	Ungültige Ethernet E/A Rack-Nummer.
701	Ungültiger Ethernet E/A Typ.
702	Zeitüberlauf bei der Ethernet-Kommunikation.
703	Allgemeiner Ethernet E/A-Fehler.
704	Der Ethernet E/A Schnittstellenanschluss wurde nicht gefunden.
705	Es konnte kein Ethernet E/A-Anschluß erzeugt werden.
706	Es konnte keine Verbindung zum Ethernet E/A Anschluss hergestellt werden.
707	Ungültige Rückmeldung vom Ethernet E/A Hauptmodul.
708	Die Option Ethernet E/A ist nicht aktiviert.
750	Die Option Kraftmessung ist nicht aktiviert.
751	Ungültiger Kraftsensor.
752	Ungültige Kraftsensor-Achse.
800	Die Sicherheitsoption ist nicht installiert.
801	Ungültiger Benutzer.
802	Ungültiges Password.
803	Zugriff verweigert.
900	Die Option Code-Lesen ist nicht aktiviert.
1000	Fehlerzähler übersteigt die maximale Anzahl von 20, die Kompilierung wurde abgebrochen.
1001	System-Fehler.

Error #	Message
1002	Falsch angezeigter Parameter.
1003	Eine Wertzuweisung mit über 100 Zeichen ist nicht zulässig
1004	System-Fehler.
1010	Eine Wertzuweisung für "&H" oder "&B" überschreitet den zulässigen Bereich. (8 bzw.32 Zeichen)
1011	Die Wertzuweisung der Real Variablen überschreitet den negativen Bereich (minus 307).
1012	Bereichs-Unter-oder Überschreitung.
1020	Zeilennummer ist zu groß.
1030	Parameter ist nicht im gültigen Bereich.
1040	Die Größe eines Array muss zwischen 1 bis 65535 sein.
1146	System-Fehler.
1147	Zeichenkettenausdruck ist nicht korrekt.
1148	Variablen-und Zeichenketten-Name sind nicht richtig.
1149	Ungültiger Funktionsname.
1150	Integer und Variablen die nicht mit "#" gekennzeichnetz sind, sind falsch.
1151	System-Fehler.
1152	System-Fehler.
1161	Zeilennummern und Label sind falsch.
1162	Ausdruck ist falsch.
1163	Fehler im Ausdruck. In Klammern setzen.
1164	Positionsausdruck ist falsch.
1170	Der Ausdruck für die Eingangsbedingung ist falsch.
1174	Punktedaten sind fehlerhaft.
1175	Angegebener Punkt ist fehlerhaft.
1176	Direkt vorgesehener Punkt ist falsch.
1177	Funktionsname ist nicht korrekt.
1178	Die Anordnung der Definitionsargumente ist falsch.
1179	Der Zeichenkettenausdruck (Pfadliste) ist falsch.
1180	Der Zeichenkettenausdruck (Verzeichnispfadliste) ist falsch.
1181	Laufwerksangabe ist falsch.
1182	Dateiname ist falsch.
1183	Verzeichnisname ist falsch.
1185	Argumentanordnung ist falsch.
1186	Parallelprozess ist nicht in Ordnung.
1187	Eine Funktion oder eine Variable im Parallelprozess kann nicht aufgerufen werden.
1188	Die Funktion mit dem Trap-Argument kann nicht aufgerufen werden.
1198	String Variable ist falsch.
1199	Der Ausdruck für den Achsenursprung ist falsch.
1200	Druckliste ist nicht richtig.
1201	Ungültiges Punkte-Label.
1231	Unzulässige Verwendung von THEN.
1244	Ungültiges CASE Format.
1254	Lokale Variablen sind falsch.
1255	Der Syntax des Arch ist falsch.
1256	System-Fehler.
1259	Klammern in der Funktion sind nicht richtig.
1260	Die Bezeichnung wurde nicht richtig beschreiben.
1261	Die Klammer ist nicht geschlossen.
1265	Der Name der Variablen ist nicht richtig.
1266	Es folgen keine Parameter "*".
1267	Ungültiges LINE INPUT Format.
1268	Ungültiges Format bei GOTO, GOSUB, CALL.
1269	Syntax Fehler.
1270	Syntax Fehler.
1400	Dieser Befehl ist im Online-Fenster nicht gültig.
1450	Funktion wurde nicht gefunden.

Error #	Message
1501	Unzulässige Verwendung von ELSE.
1502	Ungültige Verwendung von CASE.
1503	Ungültige Verwendung von EXIT.
1504	Zwischen SELECT und dem ersten CASE sind keine Anweisungen erlaubt.
1505	Block-Anweisungsfehler.
1551	Ungültige Verwendung von FEND.
1552	Anweisungen außerhalb der Funktion sind nicht zulässig.
1570	Falsche Anzahl von Parametern.
1571	Die Nummer der Positionsparameter sind nicht korrekt.
1590	Syntax Fehler bei der Macro-Anweisung.
1591	Die Macro-Anweisung beinhaltet einen Syntax Fehler.
1598	Syntax Fehler.
1600	Keine entsprechende ENDIF Anweisung gefunden.
1601	Keine entsprechende END SELECT Anweisung gefunden.
1602	Keine entsprechende WEND Anweisung gefunden.
1603	Keine entsprechende LOOP WHILE, UNTIL Anweisung gefunden.Keine entsprechende END SELECT Anweisung gefunden.
1604	Keine entsprechende LOOP Anweisung gefunden.
1605	Keine entsprechende NEXT Anweisung gefunden.
1606	Keine entsprechende IF Anweisung gefunden.
1607	Keine entsprechende SELECT Anweisung gefunden.
1608	Keine entsprechende WHILE Anweisung gefunden.
1609	Keine entsprechende LOOP WHILE, UNTIL Anweisung gefunden.
1610	Keine entsprechende DO Anweisung gefunden.
1611	Es existiert keine entsprechende FOR Anweisung.
1650	Undefinierte Zeilennummer.
1652	Undefinierte Variable.
1700	Zweimaliges ausführen von [EXIT] in ein-und derselben Verschachtelung.
1701	ELSE wurde zweimal in einer IF Anweisung gefunden.
1702	Default wurde zweimal in einer SELECT Anweisung gefunden.
1703	Die Interation hat zu viele Verschachtelungsebenen. (Maximum 16)
1704	Die bedingte Anweisung hat zu viele verschachtelte Ebenen. (Maximum 16)
1705	Zu viele Bedingungen für eine SELECT Anweisung.
1706	Die bedingte Anweisungen sind zu groß. (Maximum 32)
1750	Zu viele Pause-Deklarationen. (Maximum 4)
1755	Zu viele "String + String" Ausdrücke. (Maximum 16)
1756	Zu viele String Funktionen. (Maximum 8)
1757	Zu viele Parameter.
1758	Zu viele String Konstanten, String Variable oder String Funktionen. (Max 17)
1760	Zu viele Parallelprozessanweisungen.
1761	Zu viele Ausgangsbefehle bei einer CURVE Anweisung. (MAX 16)
1800	Der eingegebene Befehl oder die Anweisung kann nicht verwendet werden.
1908	Dieselbe Zeilennummer wurde mehrfach vergeben.
1909	Derselbe Label-Name wurde mehrmals vergeben.
1910	Derselbe Variablenname wurde mehrmals definiert.
1912	Zu viele Buchstaben.
1913	Zu viele Zeichen in einer Zeile verwendet.
1914	Zu viele Variablen in einer Zeile verwendet.
1920	Derselbe Funktionsname wurde mehrfach definiert.
1921	Derselbe Name einer externen Funktion wurde mehrmals definiert.
1930	Das Dateiende ist abnormal.
1940	Undefinierte Variable.
1941	Variable wurde nicht als ARRAY definiert.
1942	Die ARRAY Größe ist falsch.
1943	Variable wurde als ARRAY definiert.
1944	Stringbereiche können nur zwei Dimensionen beinhalten.
1950	Unbestimmter interner Funktionscode.

Error #	Message
1960	Fehlen des Arbeitsbereichs.
1970	System-Fehler.
1971	Kann internen Code nicht bestimmen.
1972	Zu viele Label in einer Datei.
1973	Zu viele Zeilen in einer Datei.
1974	Zu viele Blöcke (Anfänge) in einer Datei.
1975	Zu viele Sprung Ziele in einer Datei.
1985	Syntax Fehler.
1990	System-Fehler.
1992	Externer Funktionsbereichs ist belegt.
1993	Der Variablenbereich kann nicht gehalten werden.
1994	Kostantentabelle ist voll.
1995	Variablenspeicher ist voll.
1996	Variablentabelle ist voll.
1997	Kann Bereiche nicht beibehalten.
1998	Interner Code-Speicher ist voll.
1999	Syntaxanalyse fehler.
2000	Der eingegebene Befehl oder die Anweisung kann nicht verwendet werden.
2001	Der eingegebene TMOUT-Wert ist falsch.
2002	Der eingegebene TMR-Wert ist falsch.
2003	Sie können für das Argument keinen negativen Wert verwenden.
2004	Sie können für WAIT keinen negativen Wert angeben.
2005	Das eingegebene Argument für die Funktion TAN() ist außerhalb des gültigen Bereichs.
2006	Die Argumente der aufgerufenen Funktion passen nicht zur Definition.
2009	Die gekennzeichenete Zeilennummer existiert nicht.
2010	Die Anzahl der Zeichen innerhalb der Zeichenkette liegt über 255.
2011	Die Fehlernummer der ERROR-Anweisung liegt außerhalb des gültigen Bereichs [1-32767].
2012	Die angegebene Tasknummer liegt außerhalb des gültigen Bereichs [1-32].
2013	Die eingegebene Achsnummer liegt außerhalb des gültigen Bereichs [1-6].
2015	Der angegebene E/A-Port existiert nicht.
2016	Der abgegebene BCD-Wert liegt außerhalb des gültigen Bereichs.
2017	Die angegebene Signalnummer für SIGNAL oder WAITSIG liegt außerhalb des gültigen Bereichs [0-511].
2018	Die angegebene Merkernummer liegt außerhalb des gültigen Bereichs [0-511].
2019	Der angegebene Merkerport liegt außerhalb des gültigen Bereichs [0-63].
2020	Zu viele Ereignisbedingungen für den WAIT Befehl.
2021	Zu viele Ereignisbedingungen für den TRAP Befehl.
2022	Der eingegebene Worttyp liegt außerhalb des gültigen Bereichs [0-65535].
2023	System-Fehler.
2024	Das angegebene Verzeichnis oder der Dateiname kann nicht gefunden werden.
2025	Die COM-Nummer liegt außerhalb des gültigen Bereichs.
2026	Division durch Null.
2027	Die DSW-Nummer liegt außerhalb des gültigen Bereichs.
2028	Der angegebene Roboter existiert nicht.
2029	Sie dürfen nicht alle Eingangsbedingungen für WAIT maskieren.
2030	Sie dürfen die Bit-Bedingung für WAIT nicht maskieren.
2032	Die Elementenanzahl in dem Array ist außerhalb des gültigen Bereichs.
2033	Die Anzahl der Arrays übersteigt der Definition.
2034	Die Gruppennummer liegt außerhalb des gültigen Bereichs [1-16].
2035	Der angegebene Dateiname kann nicht gefunden werden.
2036	Die Anzahl, der in der Punktedatei definierten Punkte, liegt über der Anzahl der maximal zulässigen Punkte.
2037	Die Definition der Koordinatenachse in der Punktedatei, entspricht nicht der Punktdatendefinition im Speicher.
2038	Der SHUTDOWN-Parameter liegt nicht im zulässigen Bereichs [0-4].

Error #	Message
2039	Der Zeitparameter für ON/OFF ist zu groß. (max. 10sec.).
2040	Ungültiges Datum/Zeit-Format.
2041	System-Fehler.
2042	Die eingegebene Zahl liegt außerhalb des zulässigen Bereichs.
2043	Die Nummer der Applikation liegt außerhalb des gültigen Bereichs.
2044	Die Nummer der Konzeptdatei liegt außerhalb des gültigen Bereichs.
2045	System-Fehler.
2046	System-Fehler.
2047	Die CONFIG-Modus-Nummer ist ungültig.
2048	Die CONFIG-Protokol-Nummer ist ungültig.
2049	Die CONFIG-Baud-Raten-Nummer ist ungültig.
2050	System-Fehler.
2051	Der eingegebene TCSPEED-Wert liegt außerhalb des gültigen Bereichs [1-100].
2052	Der eingegebene HTASKTYPE-Wert liegt außerhalb des gültigen Bereichs [0-3].
2053	Das angegebene E/A-Label ist nicht definiert.
2054	System-Fehler.
2055	System-Fehler.
2056	System-Fehler.
2057	Das eingegebene Positionslabel ist nicht definiert.
2058	Punktenummer ist ungültig.
2059	Die eingegebene Ausrichtungsnummer für PORIENT ist ungültig [0, 1 oder 3].
2060	Die eingegebene LOCAL-Nummer für PLOCAL liegt außerhalb des festgelegten Bereichs [0-15].
2061	Die eingegebenen Positionsdaten für INPUT haben ein ungültiges Format.
2066	Überlauf.
2067	Negativer Überlauf.
2102	Es gibt keinen Eintrag für SPEL NT in der Registrierung.
2200	Fehler im Steuerungs-menue.
2201	Alle Bedingungen während des ausgewählten WAIT-Status sind FALSE.
2202	RCVMSG wurde bereits ausgeführt.
2203	SNDMSG wurde bereits ausgeführt.
2204	Der DECLARE-Befehl wurde noch nicht ausgeführt.
2300	System-Fehler.
2301	Die Anzahl der auszuführenden Tasks liegt außerhalb der Anzahl der maximal ausführbaren Tasks [32].
2302	Die angegebene Funktion kann nicht gefunden werden.
2303	Die angegebene Funktion wurde durch einen Ausführungsfehler angehalten.
2304	Die angegebene Funktion wird ausgeführt.
2305	Kein freier Speicher im Programm mehr vorhanden.
2308	Die angegebene Task (oder das Untersystem) kann nicht gefunden werden.
2309	Die angegebene Funktion wird editiert und kann nicht ausgeführt werden.
2310	Die Anzahl der Breakpoints außerhalb des Limits [64].
2311	Der Speicherbereich für die Stringvariablen ist nicht ausreichend.
2312	Die angegebene Task kann nicht gefunden werden.
2313	Alle CTRs laufen.
2314	Es gibt keine Bitnummer, die dem angegebenen CTR zugewiesen wurde.
2316	Das angegebene Byte beinhaltet Bits, die bereits für die Remotesteuerung zugewiesen wurden.
2317	Ungültiger INBCD-Wert.
2318	Es können nicht mehr als 4 TRAPs gleichzeitig verwendet werden.
2319	System-Fehler.
2320	System-Fehler.
2321	Meldungsfehler.
2322	Meldung überschreitet Puffergrenze.
2323	Der angegebene Roboter führt bereits eine Parallelbearbeitung aus.
2324	Während der Parallelbearbeitung trat ein Fehler auf.
2325	Der festgelegte Pfad (Datei) wurde nicht gefunden.

Error #	Message
2327	Der angegebene Task wird verwendet.
2328	Die angegebene Tasknummer liegt außerhalb des gültigen Bereichs [1-32].
2329	GOSUB-Anweisung kann im TRAP-Vorgang nicht verwendet werden.
2330	CALL-Anweisung kann im TRAP-Vorgang nicht verwendet werden.
2331	Argumente können in einer TRAP-Funktion nicht verwendet werden.
2333	System-Fehler.
2334	Parameter-Stack-Überlauf.
2335	Die angegebene DLL-Datei wurde nicht gefunden.
2336	Die angegebene Funktion wurde bereits definiert.
2337	Nicht mehr genügend Platz vorhanden für DLLFAT.
2338	Die angegebene Kommunikations-Schnittstellennummer kann nicht gefunden werden.
2339	Die angegebene Gruppe kann nicht gefunden werden.
2340	Die angegebene Gruppe kann nicht gefunden werden.
2341	Hauptfunktion wurde nicht deklariert.
2342	Hauptfunktion wurde nicht deklariert.
2343	Ausgangsbefehl kann während eines NOT-AUS-Zustands nicht ausgeführt werden.
2344	Taskbearbeitungs-Befehl kann während eines NOT-AUS-Zustands nicht ausgeführt werden.
2345	Der festgelegte Befehl kann nicht in einer bevorrechtigten Task ausgeführt werden.
2346	EResume-Anweisung kann ohne OnErr GoTo nicht verwendet werden.
2347	String-Stapelüberlauf.
2348	XQT-Anweisung kann in einem TRAP-Prozess nicht angewendet werden.
2349	Signal für WAITSIG wurde während der Überwachungszeit, definiert durch TMOUT, nicht empfangen.
2351	CALL-Anweisung kann nicht im Online-Fenster verwendet werden.
2352	Zu viele Punkte für den Curve-Pfad definiert.
2353	Der angegebene Dateiname ist zu lang.
2354	Zu viele Punkte im Curve-Pfad angegeben.
2355	Zu viele On/ Off-(Ein/ Aus) Befehle für die CURVE-Anweisung definiert.
2356	Die Größe des internen Codes für CURVE ist zu lang.
2357	Kann den Speicher für Curve nicht erhalten.
2358	Kann den Speicher für CVMove nicht beibehalten.
2359	Die angegebene Datei ist keine Kurvendatei.
2360	Die Version der Kurvendatei ist falsch.
2361	Die Manipulatornummer der Kurvendatei ist falsch.
2362	Die Größe der Kurvendatei ist falsch.
2363	Die Prüfsumme der Kurvendatei ist falsch.
2364	Interner Code ist falsch.
2401	FEND-Systemfehler.
2501	Es gibt mehr verschachtelte Funktionsaufrufe als erlaubt [16].
2502	Es gibt mehr verschachtelte FOR-Schleifen als erlaubt [16].
2503	Es gibt mehr verschachtelte IF-Verzweigungen als erlaubt [16].
2505	Es gibt mehr verschachtelte SELECT-Strukturen als erlaubt [16].
2506	Es gibt mehr verschachtelte DO-Schleifen als erlaubt [16].
2507	Es gibt mehr verschachtelte WHILE-Schleifen als erlaubt [16].
2510	Variablentyp nicht erkannt.
2520	Zu viele globale Variablen oder der Speicher für die globalen Variablen ist erschöpft.
2521	Zu viele Backup-Variablen oder der Speicher für die Backup-Variablen ist erschöpft.
2522	Die angegebene globale Variable wurde nicht deklariert.
2523	Die angegebene Backup-Variable wurde nicht deklariert.
2524	Die Gesamtgröße aller lokalen Variablen in dieser Task übersteigt das Limit.
2525	Die globale Variable wurde bereits deklariert.

Error #	Message
2526	Die angegebene globale Variable wurde bereits mit einem anderen Typ deklariert.
2527	Die angegebene Backup-Variable wurde bereits mit einem anderen Typ deklariert.
2528	Es wurde bereits eine andere Backup-Variable mit demselben Namen deklariert.
2601	Parameter-Typ-Fehler.
2602	Eine entsprechende IF-Anweisung für eine ELSE-Anweisung wurde nicht gefunden.
2603	Eine entsprechende IF-Anweisung für eine ELSEIF-Anweisung wurde nicht gefunden.
2604	Eine entsprechende IF-Anweisung für eine ENDIF-Anweisung wurde nicht gefunden.
2605	Eine entsprechende FOR-Anweisung für eine NEXT-Anweisung wurde nicht gefunden.
2606	Eine entsprechende FOR-Anweisung für eine EXIT FOR-Anweisung wurde nicht gefunden.
2607	Eine entsprechende DO-Anweisung für eine EXIT DO-Anweisung wurde nicht gefunden.
2608	Sie können nur einen String für eine String-Variable eingeben.
2609	Der festgelegte Parameter ist keine String-Variable.
2610	Ein "erwarteter" Parameter wurde angegeben.
2611	Der Typ des CASE-Eintrags passt nicht zum Variablentyp von SELECT.
2612	Eine entsprechende SELECT-Anweisung für eine CASE-Anweisung wurde nicht gefunden.
2614	In einer DECLARE-Anweisung muss dem "#" ein nummerischer Wert folgen.
2700	Die angegebene Dateinummer ist außerhalb des gültigen Bereichs [30-63].
2701	Die angegebene Dateinummer wird bereits verwendet.
2702	Die angegebene Datei wird bereits in einer anderen Task verwendet.
2703	Dateinummer nicht verfügbar.
2705	Die angegebene Datei kann nicht geschlossen werden.
2708	Datei-Suchfehler.
2709	Die angegebene Datei wurde nicht mit ROPEN geöffnet.
2710	Die angegebene Datei wurde nicht mit WOPEN geöffnet.
2711	Die angegebene Datei wurde nicht mit AOPEN geöffnet.
2712	Der angegebene Pfadname (Dateiname) ist zu lang.
2713	Ein ungültiger Pfadname (Dateiname) wurde angegeben.
2715	Es wurde eine Read-Only-Datei (nur Lesezugriff) angegeben.
2716	Der angegebene Ordner ist nicht leer oder er existiert nicht.
2717	Die Datei kann nicht kopiert werden (Pfad- oder Diskettenfehler).
2718	Die angegebene Datei wurde nicht geöffnet.
2800	Die angegebene Kommunikationsschnittstelle ist nicht verfügbar.
2801	Die angegebene Kommunikationsschnittstelle wird in einer anderen Task verwendet.
2802	Die Daten können nicht auf die angegebene Kommunikationsschnittstelle geschrieben werden.
2803	Die Daten können nicht von der angegebene Kommunikationsschnittstelle gelesen werden.
2804	Zeitüberlauf beim Lesen der Daten von der festgelegten Kommunikationsschnittstelle.
2805	Die angegebene Kommunikationsschnittstelle wird in einer anderen Task verwendet.
2806	Fehler beim Öffnen der Kommunikationsschnittstelle.
2807	Die angegebene Kommunikationsschnittstelle wurde nicht installiert.
2808	Die angegebene Kommunikationsschnittstelle wird von einer anderen Applikation verwendet.
2809	Die angegebene Einstellung wird von der Kommunikationsschnittstelle nicht unterstützt.
2830	Die angegebene Netzwerk-Schnittstelle kann nicht verwendet werden.
2831	Die angegebene Netzwerk-Schnittstelle wird in einer anderen Task verwendet.
2832	Die Daten können nicht auf die angegebene Netzwerk-Schnittstelle geschrieben

Error #	Message
	werden.
2833	Die Daten können nicht von der angegebenen Netzwerk-Schnittstelle gelesen werden.
2834	Zeitüberlauf beim Lesen der Daten von der angegebenen Netzwerk-Schnittstelle.
2835	Die angegebene Netzwerk-Schnittstelle wird in einer anderen Task verwendet.
2836	Fehler beim Öffnen der Netzwerk-Schnittstelle.
2837	Verbindung konnte nicht aufgebaut werden.
2838	Die Netzwerkeinstellungen können nicht verändert werden, während die Netzwerk-Schnittstelle verwendet wird.
2839	Zeitüberlauf für WAITCOM an der angegebenen Schnittstelle.
2840	Server oder Client erwartet.
2841	Netzwerkoption wurde nicht installiert.
2851	Die angegebene Kommunikationsschnittstelle ist ungültig.
2852	Die gewählte Baudrate ist ungültig.
2853	Die gewählte Daten-Bit-Länge ist ungültig.
2854	Das gewählte Stoppbit ist ungültig.
2855	Die gewählte Parität ist ungültig.
2856	Der gewählte Begrenzer ist ungültig.
2857	System-Fehler.
2858	System-Fehler.
2859	Der angegebene Zeitüberlauf liegt außerhalb des Nennbereichs.
2860	Die angegebene Schnittstellennummer ist nicht für die Netzwerk-Schnittstelle.
2861	Der angegebene Hostname ist ungültig.
2862	Die angegebene Schnittstellennummer ist nicht für die Netzwerk-Schnittstelle.
2901	Speicherzuweisungsfehler für den Bereich der lokalen Variablen.
2902	Fehler beim Ausführen des Interpreters.
2903	Fehler beim Freigeben des Bereichs für die lokalen Variablen.
2904	Fehler beim Zuweisen des Arbeitsbereiches für Stringoperationen.
2905	Fehler beim Freigeben des Arbeitsbereiches für Stringoperationen.
2906	Die Registry kann nicht geöffnet werden.
2907	Fehler beim Lesen von Daten aus der Registry.
2908	Fehler beim Schreiben von Daten in die Registry.
2909	Roboterfehler.
2910	Fehler am Roboter aufgetreten.
3000	Die angegebene Achsennummer ist außerhalb des gültigen Bereichs [1-6].
3001	Der angegebene Wert überschreitet die obere Grenze des Parameters.
3002	Der angegebene Wert unterschreitet die untere Grenze des Parameters.
3003	Ein ungültiger Wert [0] wurde angegeben.
3004	Ungültiger Wert. (Ein positiver Wert kann nicht angegeben werden.)
3005	Ungültiger Wert. (Ein negativer Wert kann nicht angegeben werden.)
3006	Die Anzahl der gegebenen Parameter ist ungültig.
3007	Direkt vorgesehene Positionsnummer ist falsch.
3008	Der angegebene Wert liegt außerhalb des gültigen Bereichs [1-6] von ACCEL( ).
3009	Der angegebene Wert liegt außerhalb des gültigen Bereichs [1-3] von SPEED( ).
3010	Der angegebene Wert liegt außerhalb des gültigen Bereichs [1-2] von ACCELS( ).
3011	Die angegebene Achsennummer liegt außerhalb des gültigen Bereichs [1-6] von FINE( ).
3012	Die angegebene Achsennummer liegt außerhalb des gültigen Bereichs [1-6] von JRANGE( ).
3013	Die angegebene Referenz-Datennummer liegt außerhalb des gültigen Bereichs [1-2] von JRANGE( ).
3014	Die angegebene Arch-Nummer ist zu klein für ARCH( ). (Gültige Arch-Nummern sind 0 bis 6).
3015	Die angebene Arch-Nummer ist zu groß für ARCH( ). (Gültige Arch-Nummern sind 0 bis 6).
3016	Die angebene Referenz-Datennummer liegt außerhalb des gültigen Bereichs [1-2] von ARCH( ).

Error #	Message
3022	Die aktuelle Position ist ohne Ausführen von MCAL nicht definiert. Führen Sie MCAL aus. Überprüfen Sie MCORDR, wenn ein Fehler auftritt.
3030	Roboternummer ist falsch.
3031	PRM-Datei ist defekt.
3032	Die Version der PRM-Datei ist nicht bekannt.
3033	Die angebene ARM-Nummer ist ungültig. (Gültiger Bereich ist von 0 bis 3).
3034	Die angebene TOOL-Nummer ist ungültig (Gültiger Bereich ist von 0 bis 3).
3100	Kein Roboter gefunden.
3101	Dieser Bewegungsbefehl wird nicht unterstützt.
3102	Der Status des Motion-Control-Moduls ist falsch.
3110	Es wurden keine Werte definiert.
3200	Der Roboter wird von einem anderen Task verwendet.
3201	Der Roboter wird von einem anderen Task verwendet.
3228	Fehler während des Ladens von MMD.
3300	Es wurde versucht einen Bewegungsbefehl auszuführen, während die Sicherheitsabschrankung aktiv (offen) war.
3301	Es wurde versucht einen Task auszuführen, während die Sicherheitsabschrankung aktiv (offen) war.
3302	Ein Ausgangsbefehl kann mit offener Sicherheitsabschrankung nicht ausgeführt werden.
3310	Fehler am ATTEND/NORMAL-Eingang. Starten Sie die Steuerung RC+ neu.
3311	Fehler am Eingang der Sicherheitsabschrankung. Starten Sie die Steuerung RC+ neu.
3312	Fehler am NOT-AUS-Eingang. Starten Sie die Steuerung RC+ neu.
4003	Kommunikationsfehler im Motion-Control-Modul.
4004	Fehler während des Wartens auf ein Ereignis im Motion-Control-Modul.
4006	Die Zielpunktposition liegt über dem angebenen LIMZ-Wert.
4008	Die aktuelle Punktposition oder der angebene LIMZ-Wert liegen außerhalb des Arbeitsbereichs.
4010	Die angebene Local Coordinate wurde nicht definiert.
4013	Interner Rechenfehler im Motion-Control-Modul.
4016	Es wurde versucht SFREE für eine nicht zulässige Achse auszuführen.
4017	Es wurde versucht MCAL/MCORG auszuführen, obwohl nicht alle Achsen aktiv waren.
4018	Kommunikationsfehler im Motion-Control-Modul.
4019	Der angebene Befehl wird mechanisch nicht unterstützt.
4021	Die durch LOCAL definierten Punktpositionen sind zu nah.
4022	Die Punktdaten für LOCAL sind ungültig.
4023	Der angegebene Befehl kann mit ausgeschalteten Motoren nicht ausgeführt werden.
4026	Kommunikationsfehler im Motion-Control-Modul.
4027	Kommunikationsfehler im Motion-Control-Modul.
4028	Kommunikationsfehler im Motion-Control-Modul.
4029	Kommunikationsfehler im Motion-Control-Modul.
4030	Überlauf bei Berechnung des RMS-Wertes des Drehmoments.
4033	Der angebene Befehl unterstützt keine Achsen des Pulse-Generating-Boards.
4034	Der angebene Befehl wird von diesem Robotertyp nicht unterstützt.
4035	Die Zielposition ist in einem CP-Befehl ohne räumliche Anordnung des Arms angegeben.
4036	Die Zielposition in einem CP-Bewegungsbefehl ist nicht verfügbar. (Die Zielposition ist zu nahe an der aktuellen Position).
4037	Die Armorientierung zwischen Start- und Zielposition ist unterschiedlich bei CP-Befehl.
4038	Zwei Punktpositionen eines ARC-Befehls liegen zu dicht beieinander.
4039	Die drei für einen ARC-Befehl angegebenen Punktpositionen liegen auf einer geraden Linie.
4040	Kommunikationsfehler mit dem Pulse-Generating-Board.
4041	Es wurde versucht, einen Bewegungsbefehl im verbotenen Bereich auszuführen.

Error #	Message
4042	Interrupt-Detektionsfehler.
4043	Der angebene Befehl wird nicht von diesem Robotertyp oder dieser Achse unterstützt.
4044	Die angebene Kurvenform wird nicht unterstützt.
4045	Der angebene Modus wird nicht unterstützt.
4046	Die Anzahl der Koordinaten liegt außerhalb des gültigen Bereichs.
4047	Nicht alle Punkte sind angegeben.
4048	Die Parallelbearbeitung wurde angegeben, bevor die Punkte bestimmt wurden.
4049	Die Anzahl der Parallelbearbeitungen liegt außerhalb des gültigen Bereichs.
4050	Ungültige Anzahl der angegebenen Punkte.
4051	LOCAL oder Orientierung sind nicht für alle Punkte der Kurve gleich.
4052	Nicht genügend Speicher zum Ausführen der Kurvendatei.
4053	Datei der Kurve konnte nicht erzeugt werden. Überprüfen Sie die Punkte.
4054	Fehler in der Kurvendatei.
4055	Ungültiger Abstand zwischen Kurvenpunkten.
4099	Servo-Fehler erkannt.
4100	Die Berechnung des aktuellen Punktes und der Pulsedaten ist fehlgeschlagen, weil ein Kommunikationsfehler im Motion-Control-Modul vorhanden ist.
4101	Die Berechnung des aktuellen Punktes und der Pulsedaten ist fehlgeschlagen, weil ein Kommunikationsfehler im Motion-Control-Modul vorhanden ist.
4102	Die Versorgungsspannung der Drive Unit ist nicht eingeschaltet.
4150	Fehler am NOT-AUS-Eingang.
4151	Fehler am Eingang der Sicherheitsabschrankung.
4180	Die angebene Roboter-Nummer ist ungültig.
4181	Der angebene Roboter wird von einem anderen Task verwendet.
4182	Robotername ist zu lang.
4183	Versionsfehler der Roboterdaten.
4184	Es ist mehr als ein Gelenk einer einzelnen Achse zugewiesen.
4185	Es ist mehr als ein Roboter einer einzelnen Achse zugewiesen.
4186	Erforderliche Hardware wurde nicht gefunden.
4187	System-Fehler.
4188	System-Fehler.
4189	System-Fehler.
4190	System-Fehler.
4191	System-Fehler.
4192	Kommunikationsfehler mit dem Servo-Modul.
4210	Der RAS-Kreislauf entdeckte eine Fehlfunktion der Steuerung.
4211	Servo-CPU interner RAM-Fehler.
4212	Drive Unit Dual-Port-RAM-Fehler.
4213	Servo-CPU Programm-RAM-Fehler.
4214	Servo-CPU-Prüfsummenfehler in der Initialisierungsanweisung.
4215	Servo-CPU-Überlauf.
4216	Servo-Echtzeit-Befehlsüberlauf.
4217	Servo-Prüfsummenfehler im Echtzeitbefehl.
4218	Servo-Langzeit-Befehlsüberlauf.
4219	Servo-Prüfsummenfehler im Langzeitbefehl.
4220	Fehler in der Zeitüberwachung der Drive Unit.
4230	Servo-Prüfsummenfehler im Echtzeitstatus.
4231	Servo-Prüfsummenfehler im Langzeitstatus.
4232	Servo-Fehler im Freilaufzähler.
4233	Servo-CPU-Kommunikationsfehler.
4240	Ungültigen Interrupt erkannt.
4241	Zu hohe Geschwindigkeit im Low Power-Modus.
4242	Ungültiger Beschleunigungswert wurde erkannt.
4243	Ungültiger Geschwindigkeitswert wurde im High-Power-Modus erkannt.
4250	Der Arm hat die Grenze des Arbeitsbereiches erreicht.
4251	Der Arm hat die Grenze des X/Y-Arbeitsbereiches erreicht (festgelegt durch

Error #	Message
	XYLIM).
4252	Der Arm hat die Grenze des Arbeitsbereiches erreicht.
4301	Das Pulse-Generating-Board hat ein Begrenzungssignal erkannt.
4302	Das Pulse-Generating-Board hat ein Alarmsignal erkannt.
5000	Fehlfunktion des Servo-Gate-Arrays.
5001	Kabeltrennung Encodersignal.
5002	Motortreiber ist nicht installiert.
5003	Zeitüberlauf bei der Initialisierung des Inkremental-Encoders.
5004	Zeitüberlauf bei der Initialisierung des Absolut-Encoders.
5005	Einstellungen für die Encoderteilung sind ungültig.
5006	Absolut-Encoder-Zeichenfehler.
5007	Absolut-Encoder-Rotationsüberlauf.
5008	Pulsezähler-Überlauf.
5016	Back-up Alarm des absoluten Encoders.
5017	Prüfsummenfehler des absoluten Encoders.
5018	Batteriefehler im absoluten Encoder.
5019	Absolut-Encoder: Absoluter Fehler.
5020	Überdrehzahl im absoluten Encoder.
5021	Absoluter Encoder ist überhitzt.
5032	Servo-Alarm A.
5040	Fehler im Motorendrehmoment im High-Power-Status.
5041	Fehler im Motorendrehmoment im LOW-Power-Modus.
5042	Positionsfehler im High-Power-Modus.
5043	Positionsfehler im Low-Power-Modus.
5044	Geschwindigkeitsfehler im High-Power-Modus.
5045	Geschwindigkeitsfehler im Low-Power-Modus.
5046	Zu hohe Geschwindigkeit im High-Power-Modus.
5047	Zu hohe Geschwindigkeit im Low-Power-Modus.
5048	Überspannung.
5049	Überstrom.
5050	Zu hohe Geschwindigkeit im Modus der Drehmomentsteuerung.
5054	Servo-Überlastung A.
5055	Servo-Überlastung B.
5072	Servo-Alarm B.
5080	Servo-Überlastung C.
5112	Servo-Alarm C.
5120	Servo-Überlastung D.
5136	Der Absolut-Encoder wurde initialisiert.
5152	Servo-Alarm D.
6001	System-Fehler.
6002	System-Fehler.
6003	System-Fehler.
6004	System-Fehler.
6005	System-Fehler.
6006	System-Fehler.
6007	System-Fehler.
6008	System-Fehler.
6102	Eingangssignal-Fehler des ATTEND/NORMAL-Schalters.
6103	Der angebene Roboter wird an der Remote-E/A nicht gefunden.
6104	Fehler am NOT-AUS-Eingang.
6500	Keine Antwort vom Optional Device.
6501	Das Optional Device ist belegt.
6502	Daten können nicht auf das Optional Device gesendet/geschrieben werden.
6503	Einstellungen für Optional Device können nicht gefunden werden.
6504	System-Fehler.
6505	COM3 ist nicht bereit.
6506	System-Fehler.

Error #	Message
6507	COM3-Anordnungsfehler (Framing-Fehler).
6508	COM3-Überlauffehler.
6509	COM3-Fehler: Empfangsspeicher voll.
6510	COM3 Parity Fehler.
6511	COM3: anderer Fehler.
6701	Fehler: TCP/IP-Schnittstelle offen.
6901	Unerwartete Version der Roboterdaten.
6902	Roboterdaten Datei-Lesefehler.
6903	Roboterdaten Datei-Schreibfehler.
6904	Roboterdaten Speicher-Zuordnungsfehler.
6905	Roboterdaten können nicht geändert werden.
7000	System-Fehler.
7103	Zu viele Zähler.
7350	Fehler beim Einrichten eines asynchronen Ausgangs.
7351	System-Fehler.
7352	System-Fehler.
7353	Fehler beim Erzeugen einer Speicher-Mapped-Datei.
7901	Fehler beim Erzeugen einer Meldungstabelle.
7902	System-Fehler.
7903	Fehler beim Erstellen des Bereiches für die Programmierung.
7904	Fehler beim Erstellen des Bereiches für das Untersystem.
7905	Fehler beim Erstellen des Bereiches für die Breakpoint-Informationsdaten.
7906	Fehler beim Erstellen des Bereiches für die Timer-Informationsdaten.
7907	Fehler beim Erstellen des Bereiches für den Steuerungsblock.
7908	Fehler beim Erstellen des Bereiches für die Roboter-Informationsdaten.
7909	Fehler beim Erstellen des Bereiches für die lokalen Variablendaten.
7910	System-Fehler.
7911	System-Fehler.
8000	Die Systempunktedatei ist ungültig.
8001	Die Systempunktedatei kann nicht erzeugt werden.
8002	Die Systempunktedatei kann nicht geschlossen werden.
8003	Die Roboterummer ist ungültig.
8004	Die Systempunktedatei wurde bereits erzeugt.
8006	Ungültige Punktnummer.
8009	Die angebene Punktedatei kann nicht geöffnet werden.
8010	Der Bereich für die Palettendefinition kann nicht erstellt werden.
8011	Der Bereich für die Palettendefinition kann nicht freigegeben werden.
8012	Die Palettennummer ist außerhalb des gültigen Bereichs.
8013	Die angebenen Palettendaten sind nicht definiert.
8014	Der Palettenteilungswert ist außerhalb des gültigen Bereichs.
8015	Koordinaten-Achsen-Fehler.
8016	Punkt-Flag ist ungültig.
8017	Eine Punkte-Label-Tabelle kann nicht erzeugt werden.
8018	Das angegebene Punkt-Label kann nicht gefunden werden.
8019	Die angegebene Systempunktedatei kann nicht gefunden werden.
8020	Punktdaten-Datei -Schreibfehler.
8021	Die angegebene Punktdatei wird bereits benutzt.
8022	Die lokalen Koordinaten der angegebenen Positionen sind nicht übereinstimmend.
8100	Das angegebene Punkt-Label wurde nicht definiert.
9081	System-Fehler.
9082	System-Fehler.
9083	Parameterfehler in der SRCLoad-Beendigungs-Funktion.
9100	System-Fehler.
9101	System-Fehler.
9102	System-Fehler.
9103	System-Fehler.
9104	System-Fehler.

Error #	Message
9201	System-Fehler.
9202	System-Fehler.
9203	System-Fehler.
9204	System-Fehler.
9205	System-Fehler.
9206	System-Fehler.
9207	System-Fehler.
9208	System-Fehler.
9209	System-Fehler.
9210	System-Fehler.
9211	System-Fehler.
9212	System-Fehler.
9213	System-Fehler.
9214	System-Fehler.
9215	System-Fehler.
9216	System-Fehler.
9217	System-Fehler.
9301	System-Fehler.
9302	System-Fehler.
9401	System-Fehler.
9402	System-Fehler.
9403	System-Fehler.
9404	System-Fehler.
9701	System-Fehler.
9702	System-Fehler.
9703	System-Fehler.
9704	System-Fehler.
9705	System-Fehler.
9706	System-Fehler.
9707	System-Fehler.
9800	System-Fehler.
9880	System-Fehler.
9901	System-Fehler.
9950	System-Fehler.

Glossar

abgeblendet

Wenn ein Menüpunkt, ein Radiobutton, eine Checkbox, ein Button usw. einen helleren Farbton hat (normalerweise grau), ist dieser Punkt nicht verfügbar.

Alphanumerisch

Ein Zeichen, das ein Buchstabe (a-z, A-Z) oder eine Zahlzeichen (0-9) ist.

ANSI

American **N**ational **S**tandards **I**nstitute. US-amerikanisches Komitee für Standards (Normung). Die ANSI/RIA 15.06 Spezifikation ist beispielsweise der US-amerikanische Standard für Roboter-Sicherheitsanforderungen.

Arbeitsbereich

Der gesamte Arbeitsbereich einer Roboter-Arbeits-Zelle. Der Arbeitsbereich ist als der Bereich definiert, welchen die Roboter-Arbeits-Zelle belegt. Der Arbeitsbereich kann in drei Teile unterteilt warden:

Der maximale Bereich – Der größte mögliche Bereich oder Rauminhalt, in welchem sich der Roboter bewegen kann.

Der begrenzte Bereich – Der Anteil des maximalen Bereiches, auf welchen der Roboter durch begrenzende Geräte begrenzt wird.

Der Betriebsbereich – Der Bereich, in dem der Roboter arbeitet.

ASCII

American **S**tandard **C**ode for **I**nformation **I**nterchange (Amerikanischer Standard-Code für Informationsaustausch) Ein weit verbreitetes numerisches System zur Kodierung von Buchstaben, Ziffern, Satzzeichen, Zeichen und anderen Zeichen wie Binärzahlen.

asynchron

Das Konzept, Zeichen zwischen einem Start- und ein Stop-Bit einzuschließen, wird als asynchrone Übertragung bezeichnet. Das Startbit signalisiert der empfangenden Seite die Zeit, wann nach den Zeichenbits Ausschau gehalten werden soll; das Stopbit lässt es die Dateneinheit wissen, wenn das gesamte Zeichen empfangen wurde. Das "timing" für die Kommunikation wird von den Start- und Stopbits bestimmt. Daher können Zeichenübermittlungen aufgrund der asynchronen Übermittlungstechnik in sporadischen (oder nicht-synchronisierten) Intervallen erfolgen (d.h. asynchron bedeutet nicht zeitlich festgelegt, nicht regelmäßig oder nicht synchronisiert).

Ausrichtung des Armes

Bei der Verwendung von SCARA-Robotern bezieht sich die Ausrichtung des Armes auf die Ausrichtung des Ellbogengelenks. Es gibt zwei mögliche Ausrichtungen: Lefty (Linksarm) oder Righty (Rechtsarm). Wenn der Ellbogen wie ein menschlicher linker Arm konfiguriert ist, wird dies Linksarm-Orientierung genannt. Wenn der Ellbogen wie ein menschlicher rechter Arm konfiguriert ist, wird dies Rechtsarm-Orientierung genannt.

Beschleunigung

Die Rate, um welche der Roboterarm zu Beginn einer bestimmten Bewegung beschleunigt (oder bis zur Nutzer-definierten Geschwindigkeit hochläuft). Die Beschleunigungsrampe wird in Strecke/Zeit² gemessen. Die Definition der Beschleunigung einer Bewegung erfolgt durch die Verwendung der Befehle Accel oder AccelS.

Binär codierte Dezimalstelle

Binärzahlen können in der Regel leicht in Hexadezimal- oder Oktalzahlen umgewandelt werden. Eine binär codierte Dezimalstelle bedeutet, eine Binärzahl zurück in eine Dezimalzahl (Basis 10) zu konvertieren. Das bedeutet, dass jede Gruppe aus 4 Binärzeichen nur in eine entsprechende Dezimalzahl zwischen 0 und 9 konvertiert werden kann. Die Werte 10 bis 15 (welche leicht als Hexadezimalzahlen dargestellt werden können) sind im BCD-Format nicht nutzbar. Binär codierte Dezimalstellen werden in der Regel verwendet, um Hardware-Eingänge in eine Basis-10-Zahl zu konvertieren, welche dann für einen Rändelradschalter oder andere Geräte verwendet werden kann, die hauptsächlich Basis-10-Zahlen verwenden.

Bogenbewegung

Bogenbewegung ist ein anderer Name für eine kreisinterpolierte Bewegung. Die kreisinterpolierte Bewegung ist ein Typ einer Kurvenbewegung, bei der ein Bogen durch 3 Punkte definiert wird. Der Arm bewegt sich kreisförmig entlang dieser 3 Punkte bis er schließlich den dritten Punkt erreicht hat. Die Bogenbewegung wird normalerweise bei Prozessanwendungen wie z.B. dem Dispensen verwendet.

Clipboard

Ein temporärer Speicherplatz, der verwendet wird, um Text und Punkte zwischen Dokumenten zu übermitteln.

Continuous Path (kontinuierlicher Weg)

Ein Continuos Path (kontinuierlicher Weg) ist ein Satz von Punkten, an denen der Roboter mit einer konstanten Geschwindigkeit entlang fährt. Mehrere Befehle in SPEL werden verwendet, um die CP-Bewegung zu unterstützen. (Arc, Pass.)

CP

Continuous Path (kontinuierlicher Weg). Eine CP-Bewegung (kontinuierlicher Weg) ist eine besondere Art der Bewegung, bei welcher der Roboter mit einer konstanten Geschwindigkeit einem vordefinierten Satz von Punkten folgt, darauf achtend, dass während der Bewegung alle Kurven geglättet werden. SPEL hat eine hohe CP-Leistungsfähigkeit, die besonders für Prozessanwendungen wie z.B. dem Dispensen hilfreich ist.

Funktion

Der Hauptteil eines Programmcodes, der als ein Task im System läuft. Siehe "Funktion” in der Befehlsreferenz für mehr Details.

Geradlinige Bewegung

Bewegt den Roboterarm in einer geraden Linie mit den aktuellen SpeedS- und AccelS-Einstellungen von der aktuellen Position zu einem spezifizierten Punkt.

Gültigkeitsbereich

Der Gültigkeitsbereich einer Variable spezifiziert, wo die Variable erkannt wird, sobald sie deklariert wurde.

kreisinterpolierte Bewegung

Die kreisinterpolierte Bewegung ist ein Typ einer Kurvenbewegung, bei welcher ein Bogen durch 3 Punkte definiert wird. Der Arm bewegt sich kreisförmig entlang dieser 3 Punkte bis er schließlich den dritten Punkt erreicht hat. Die kreisinterpolierte Bewegung wird normalerweise bei Prozessanwendungen wie z.B. dem Dispensen verwendet.

Point to Point-Bewegung (PTP-Bewegung)

Bewegt den Roboterarm in kürzest möglicher Zeit mit den aktuellen Geschwindigkeits- und Accel-Einstellungen von der aktuellen Position zu einem spezifizierten Punkt. Diese Bewegung kann geradlinig erfolgen, muss es aber nicht.

Programmgruppe

Eine Programmgruppe beinhaltet ein oder mehrere SPEL-Programm(e). Eine Programmgruppe kann mit dem Chain-Befehl innherhalb eines Programms ausgeführt werden. Programmgruppen sind optional.

RCF-Datei

Eine für jedes Projekt automatisch erzeugte Datei, welche die Informationen zur der Systemkonfiguration enthält, einschließlich Hordr, Weight, TLSet usw.

RIA

Robotics **I**ndustries **A**ssociation, US-amerikanischer Robotikverband. Organisation von Roboterverkäufern und Endverbrauchern, deren Zweck die Förderung des Verständnissen von Robotern und des Robotergebrauches ist.

Roboterpunkt

Eine kartesische Koordinate, die eine Roboterposition im Arbeitsbereich spezifiziert. Jeder Punkt hat 4 Koordinaten: X, Y, Z, U und zwei Attribute: Ausrichtung (lefty, righty) und Local-Nummer (0-15). Die Punkte werden in Projekt-Punktdateien auf dem PC gespeichert.

Task

Eine Objektdatei, die im System ausgeführt wird. Wenn in SPEL eine Funktion gestartet wird, welche die Xqt-Anweisung verwendet, wird diese als separater Task ausgeführt. Bis zu 32 Tasks können zur selben Zeit in der Robotersteuerung laufen.

Tool (Werkzeug)

Eine Vorrichtung oder ein End-Effektor, der am Handgelenk des Roboterarms montiert ist. Ort und Ausrichtung des Tools mit dem TLSet-Befehl wird von Ihnen definiert.

Verzögerung

Die Rate, um welche der Roboterarm am Ende einer Bewegung verzögert (oder verzögert, wenn er an einen Stopp kommt). Die Verzögerungsrampe wird in Strecke/Zeit² gemessen. Die Definition der Verzögerung einer Bewegung erfolgt durch die Verwendung des Befehls Accel oder AccelS.